TAMPERE UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF PERVASIVE COMPUTING

SACHIN RAJ MISHRA

A REVIEW FRAMEWORK FOR OPEN SOURCE ORIENTED SOFTWARE

MASTERS OF SCIENCE THESIS

Topic Approved by:

Faculty Council of

Computing and Electrical Engineering on

May 2013

Examiners:

Adj. Professor Dr. Tech Imed Hammouda

Researcher Mr. Alexander Lokhman

Abstract

TAMPERE UNIVERSITY OF TECHNOLOGY

Master's Degree Program in Information Technology

MISHRA, SACHIN RAJ: Software Quality Review Framework for Open Source Software

Master of Science Thesis, 68 Pages,

September 2012

Major subject: Software Systems

Examiner(s): Professor, Dr. Tech. Imed Hammouda, Researcher Mr. Alexander Lokhman

Keywords: Open Source Software (OSS), Quality Assurance (QA), Community, Licensing, quality attributes, quality factors

Software Quality Assurance is an essential yet challenging process which consists of several milestones. There exist several Quality assurance models and frameworks (both fixed and flexible) for reviewing software of any type. Fixed models consist of fixed set of quality attributes and their measures, whereas for the flexible model the attributes are decided or chosen based on requirement set of the product. Earliest models like McCall's, Boehm's, FURPS and ISO/IEC 9126 are examples of fixed models. Whereas, Prometheus model developed in 2003 is an example of flexible model. It means, ever since 1977, there have been quite a lot of QA models, frameworks and standards published, in order to ease the vigorous process of QA. Most of these models are product-centric. Most of the product-centric QA models are the derived work of McCall's model, Boehm's model, Garvin's model, FURPS framework and ISO/IEC 9126 standard in one way or another (in lower or higher degree). Hence, these primitive models are somehow the base models. For several reasons, not all the base models are completely applicable; not at least to Open Source Software (OSS). OSS is a movement or a philosophy where software and its binary are freely available to everyone allowing modification or redistribution.

There are 3 major dimensions through which OSS could be observed; as a Community, as a Licensing model and as a development Method. There are several widely adopted trends followed in typical OSS development. One of which is to present a mature enough product to a community and ask them to contribute in different ways. Here the mature product includes a set of initial design, deliverables including requirements specifications and available source code (if any). In this typical trend, the community members are geographically diverse or distributed. In contrary to the typical development setting there exists a varied development setting of OSS. In this setting, the development starts and continues as in-house project by a small group of core developers who were

solely responsible for designing the software, choosing the development settings, choosing the licenses, implementing, doing the market research, testing the software, registering it on public forge and finally releasing the software. The software is, at the end, publicized to the open source community as OSS. The initial development does not include anyone else than the core developers. These core developers are not geographically diverse. These core developers or the project team uniquely owns the right for the initial state of the software. For these variations we call this type of software Open Source Oriented Software (OSOS). There subsist some differences; therefore, the available QA models for OSS are not completely applicable for OSOS.

In order to fill this gap, we propose a framework which could be used to review software adopting OSOS development setting. We called this framework LCM framework. The reason behind the name is the three aforementioned perspectives towards OSS namely Licensing, Community and Method. In order to attain this framework, the base models are comprehensively analyzed towards our requirements. LCM framework consists of quality attributes and sub-attributes as the measures. These attributes are then categorized as Community Compliance Attributes, Licensing Compliance Attributes and Method Compliance Attributes.

In order to assure the result, LCM framework was used over OSOS named Solution to Open Land Administration (SOLA) developed by United Nation Food and Agriculture Organization. Four different versions of SOLA application were reviewed using the LCM framework. The results encountered for each review helped improve the quality of later versions of SOLA application.

The results of SOLA review are divided into three parts; behavioral analysis results for, Community Compliance Attributes, Licensing Compliance Attributes and Method Compliance Attributes. Static analysis (code analysis) on the other hand was the basis of comparison for most of the behavioral analysis results for Community Compliance Attributes and Licensing Compliance Attributes. The static review was performed based on the data collected by Sonar, which is an open source quality management platform, dedicated to measure source code quality.

The LCM framework when used over an open source project yield improving results. Therefore, it could be said that LCM framework is adoptable to all the software developed with OSOS development setting. However, the choice of attributes according to the stage of development is different for software with different requirements.

Preface

This Master of Science Thesis has been carried out in the Department of Pervasive Computing at Tampere University of Technology (TUT), Tampere, Finland as a part of Solution for Open Land Administration (SOLA) project by United Nation Food and Agricultural Organization (UN FAO) during June 2011 – April 2012. The thesis work has been funded as an ongoing research project at the department.

I am pleased to express my gratitude and appreciation to my thesis supervisors, Professor Dr. Imed Hammouda and Mr. Alexander Lokhman for their valuable guidance and support throughout the thesis period. Their way of sharing knowledge and willing to help attitude has supported me a lot to stand-in my work in the correct direction. I am also grateful to Mr. Andrew McDowell and Mr. Neil Pullar from UN FAO for their valuable comments and suggestions regarding the results.

Finally, it is a pleasure to thank my family and friends for their support and motivation in order to pursue this Master's degree.

Tampere, 28th March, 2013

Sachin Mishra

Contents

List of Abbreviations

Abbreviations	Descriptions
OSS	Open Source Software
SDLC	Software Development Life Cycle
OSOS	Open Source Oriented Software
OOP	Object Oriented Programming
QA	Quality Assurance
SOLA	Solution for Open Land Administration
GPL	General Public License
LGPL	Lesser General Public License
BSD	Berkeley Software Distribution
MIT	Massachusetts Institute of Technology License
MozPL	Mozilla Public License
UN FAO	United Nation, Food and Agricultural Organization
FURPS	Functionality, Usability, Reliability, Performance, Supportability Model
OSI	Open Source Initiative
ISO	International Organization for Standardization
IEC	International Electro-technical Commission
FSF	Free Software Foundation
OSSD	Open Source Software Development
LCM	Licensing, Community, and Method Quality Assurance Model

Table of Tables

Table of Figures

1. Introduction

"Quality in a service or product is not what you put into it. It is what the client or customer gets out of it." – Peter Drucker. This statement could not be ignored. However, there lies a possibility where this statement could be polished. It could be argued that if the customers will get a quality service and product, it is due to the reason that quality is put into it. In order to put quality in any product, one needs to intensely identify and analyze the requirements from various level of the product development. Product development usually starts from the initial market study till the final product support with various requirements. Based on these requirements, the implementation has to be made. Also the product has to be evaluated over different measures of quality, known as quality attributes/ quality factors/ quality characteristics. There are several factors, by the help of which the quality could be enhanced. As in general, the definition of quality for any service or a product is similar. However, the way of evaluating the quality for products depends on the requirement of its customers as well as the product itself.

1.1. Motivation

Alike many products, software needs to verify its quality. Software quality is a major concern for different types of software. There are usually 2 wide ranges of software category including closed source software and Open Source Software (OSS) with their own differences. One of the differences is the requirements set for the software. In a typical OSS development, it is hard to use traditional development model like waterfall. The reason behind it is the requirements, which are not known beforehand. Another difference is the development team which is mostly found distributed in OSS development. In addition, the ownership of OSS does not lie on of a person or a company. Instead, it is free and for all who wish to hold it.

As it is known, in a typical OSS development setting, the software is meant to be OSS from the very beginning. It follows the typical open source development method which includes proper choice of communication channels, tools, methodologies to follow and exposure to the community. The community, which then acts as the core of OSS, holds responsibility to test the software in order to assure software quality. Keeping aside the typical OSS development setting, we think about a variation, which is indeed possible. Here the software project is not exposed to any community from the beginning. It is developed by a bunch of developers (not distributed) who uniquely own the right for the software. However, it is kept in the mind that the final software is to be released as OSS. Processes such as registration, communication, marketing and tools are solely decided by these developers. In this development setting, the end product is OSS but the process however diverts from a typical OSS development approach. We, therefore, call this kind of software Open Source Oriented Software (OSOS). It certainly seems to be different than commonly developed OSS in many regards.

In this work, our major concern is software quality. The software quality is a process of evaluating software from different perspectives. The outcomes of these evaluations are then reported and the enhancement or change is made according to these reports. One of the several product evaluation processes is review. For different variety of product and product category the review process varies in a higher degree. For example, software products have various review models, quality models, frameworks and standards present. These models are used to assure and deliver quality software end product to the public. This work is further narrowed down the software category and its development method OSS being one of them. OSS is indeed one of the emerging as well as competitive methods. It has several quality models for quality assurance.

1.2. Objective

As mentioned earlier, in our context, even though the final product is OSS, there are some diversions in the development process and settings, yielding OSOS. The study shows that there is no complete quality assurance frameworks available for this type of software developed. Due to this reason, the main objective of this thesis work is to propose a review framework for software which has adopted similar development settings as that of OSOS.

To present the final review framework for OSOS, a comparative and analytical review methodology was take-on for different quality and review models that have been published since 1977, for example McCal's Quality Assurance (QA) model, Boehm's QA model and so on. All of the available models comprised of different quality attribute. Some of these quality attributes were adopted as they were found relevant in our context. Some of them were removed as being irrelevant. On the other hand, the missing ones were added forming final framework. Keeping in mind that the end product is to be released as OSS the relevance and irrelevance of the quality attributes were chosen based on the OSS requirements and perspectives including community requirement, licensing requirement and development method requirement.

The final product consists of 17 quality attributes. These attributes are categorized under three dimensions. The proposed framework was then used over an open source project called Free/Libre Open Source Software Solution of Open Land Administration (FLOSS SOLA). There was four different review made on four different versions of the application. Each review report yielded in better versions. The proposed review framework was named Licensing, Community and Method (LCM) framework. Some of the quality attributes that are included in the LCM framework are Reliability, Maintainability, Performance, Accessibility, Security, Usability, Portability, Trademark, Copyright, Development Infrastructures, Software Marketing and Product Registration and so on.

1.3. Organization

The organization of this thesis is as follows:

Chapter 1 provides a broader overview of this thesis. The motivation for this work followed by its objective is presented in this chapter. In Chapter 2, detailed discussion about what is open source software and software quality assurance is made. Furthermore, major perceptions on open source software are discussed in this chapter. Some strong and weak aspects of different existing quality assurance models are evaluated. The evaluation is made based on their contents and attributes used to form these models. In addition, a comparison based table of these models is presented and our resulting framework is proposed.

Chapter 3 presents a detailed overview and insights on the context where the proposed framework could be used. In sub sections, the proposed framework is further refined based on the relevance and interpretation of each available quality attributes. How the categorization was made for each quality attributes are discussed in this chapter.

Chapter 4 contains the results from the case study where the proposed framework was used. The categorization and interpretation of each achieved results are discussed in this chapter. We also revisit our purpose in the discussion section of this chapter.

Finally in chapter 5, the thesis is concluded with limitations that we faced and further ideas that could be used as a future work.

2. Open Source Software and Quality Assurance

In this chapter detailed discussion about what is open source software and software quality assurance is made. Furthermore, major perceptions on open source software are discussed in this chapter. Some strong and weak aspects of different existing quality assurance models and their differences are evaluated. The evaluation is made based on their contents, attributes and sub-attributes used to form these models. In addition, a comparison based table of these models is presented and our resulting framework is proposed.

2.1. Open Source Software

In 1983, a movement was started and lead by a computer scientist Richard Stallman which later took a shape of a foundation named FSF. FSF since then have been providing their definitions on what is free software? According to FSF, free software is the software which provides user a freedom to run, copy, modify, study, distribute, change and improve the software. As an important matter, FSF clarifies the concept of freedom as in liberty rather than in price. Hence, it came across 4 different freedoms that are essential for any software to be free software, which are, (0) a freedom to run, (1) a freedom to study, (2) a freedom to redistribute and (3) a freedom to distribute copies of the modified versions out of which freedom 1 and 3 have the precondition of accessible source code.

According to the OSI, software that is freely redistributable, modifiable and which is not privately owned is OSS. In addition, any software that meets following requirements set by OSI could be labeled as an OSS.

- Free Redistribution
- Source Code
- Derived Works
- Integrity of The Author's Source Code
- No Discrimination Against Persons or Groups
- No Discrimination Against Fields of Endeavor
- Distribution of License
- License Must Not Be Specific to a Product
- License Must Not Restrict Other Software
- License Must Be Technology-Neutral

OSS is first of the software kind [2] which is developed in late 70’s. OSS was dominated by the proprietary software in early 80’s. There have been arguments between Free Software and Open Source Software. Here in this thesis we will not discuss or differentiate between them but rather call it as Free\ Libre Open Source Software (F\LOSS) (referred as OSS in later sections).

OSS is free software which is accessible to everyone. It provides right of distribution of licenses and which could be adopted as a framework for developing software. Hence,

there are at least three major perspectives around which OSS could be defined. These perspectives or the influential factors are depicted in figure 2.1 following the detailed description in later sections.

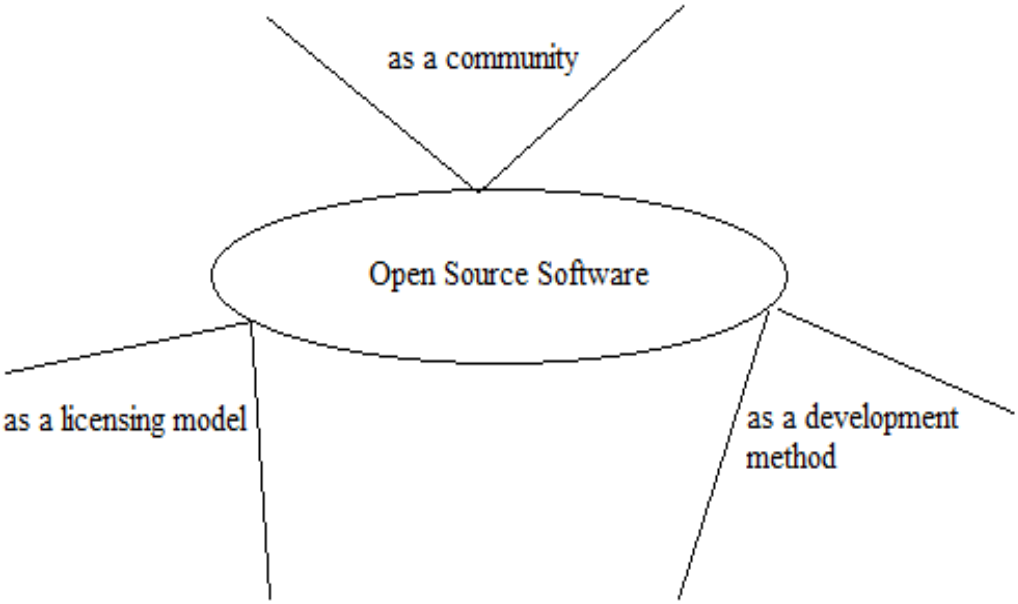


Figure 2.1 Perspective towards Open Source

2.1.1. Open Source as a Community

The idea of development of OSS is generally triggered by the personal itch [16]. The concept is then put forward to the public, with the expectation of contribution in forms of development, review, support and use. These interested personnel then take the idea and start implementing it; same or different group of people then performs the review and finally it is made available for public use. In this overall process all the people involved in developing this idea are normally distributed, but are connected via some communication media (mostly internet). They work on the same idea following same conventions. These groups of people in the context of Open Source are known as open source community members. Almost all the OSS has its own community or is merged to some preexisting ones. One of the largest open source communities is Linux community which contains over a million of developers, users and other contributors from around the world. Even the world's largest Open Source Software would have been a failure without the contribution of users and the developers from the community. [16] Therefore the key to success is to consider community as a major perspective in the development of OSSs.

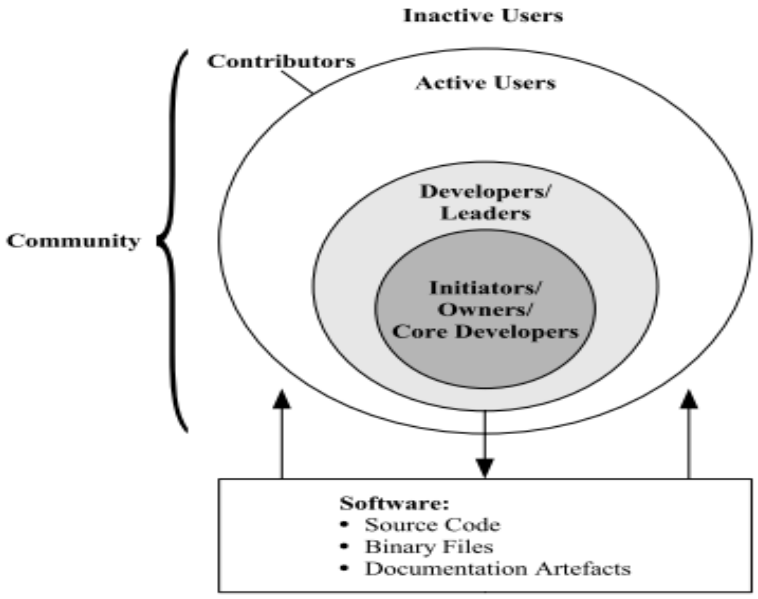


Figure 2.2 In-depth community structure for an open source project [17]

Figure 2.2 is a typical onion model for Open Source Software community structure. This structure is a layered structure which in core contains the initiators, circled around other developers and leaders. These two layers are the developer community whereas the top most two layers are for the users both active and inactive and hence is a user community. However, there is no restriction towards developers using the software and users contributing as a developer. This overall structure could also be known as an onion structure for the typical open source community.

2.1.2. Open Source as a Licensing Model

Similarly, OSS could be used as a licensing model. Licensing model in the sense that OSS has to follow a different set of agreements for redistributions and restrictions. Speaking of the OSS definition by OSI, the distribution term for OSS must comply with certain criteria and these criteria must be clearly mentioned for each module of open source software.

The license shall not restrict on sharing or distributing the software and its components and it shall not require a royalty or other fee for such distribution. Furthermore, the license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software. The license must allow distribution of software built from modified source code. However, the license may require derived works to carry a different name or version number than that of the original software. In addition, the license must not discriminate against any person, group of persons or field of endeavor [Section 0]. The license must not place restrictions on other software that is distributed along with the licensed software and no provision of the license may be predicated on any individual technology or style of interface. These criteria may be listed in any order to form a different type of license model for OSS. There can be some more flexibility towards the terms in each license model.

Few examples of Open Source Licenses are GNU GPL, GNU LGPL, MIT Licenses, Apache v1, v1.1 and v2, CPL, CDDL, Educational Community License (ECL), BSD and modified BSD.

GNU GPL license version 2, June 1991 FSF contains following information:

Copyright © 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users…… .

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program

2. You may copy and distribute…

3. You may modify your copy or copies…

4. …

…

10.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSES FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL AND COPYRIGHT HOLDER…

END OF TERMS AND CONDITIONS

Figure 2.3 GNU GPL License version 2

2.1.3. Open Source as a Development Method

Software development is a human activity with multiple planes which could be analyzed from different viewpoints. [5] All these planes and viewpoints are however minimized and solved by answering only two questions: what and how. In any software development method what questions and how questions may appear in following ways:

- What is required? How to acquire it?
- What are the problems? How to get the solutions?
- What to describe? How to describe it?
- What are the requirements and specifications? How to develop and integrate them?

These questions during the development of any software remain same. However, based on the nature of software the answers may differ. A single development methodology/ framework may not be applicable for the entire software kind. As mentioned earlier, OSS is a philosophy and movement. It is also a recurring development framework/ method. It could be used to structure, plan and control the process of development. The structuring and planning is done by predefining the milestones such as deliverables and artifacts.

There are several ways to develop OSS. One can initiate the project or present the idea to the public and ask them for help in developing that idea furthermore. The same can contribute on an existing product or fork a well-established product and make a parallel development. Apart from these it is also possible to develop OSS in-house. When the software is mature enough, the version is put or released to a community. No matter which way one follow, all need a set of process-data model.

For general software, the development starts by analyzing a problem, doing market research and gathering requirements for the proposed business solution and then finally the design plan for software based solution. The right methodology to adopt is then decided and implementation, testing, deployment and maintenance are followed in contrary to OSS development method. The open source development method mostly concern appropriate choices. Choices for methodologies, for example, OSSs mostly avoid waterfall model (due to unfixed requirements), choices for the right development tools are most of a concern. The development tools which are considered most important are chosen. These tools include use of proper communication channels, bug tracking tools, version controls, testing tools and package management tools. Setting up the common development methodology for revamps and rewriting of the codes is decided. Building a community and publicizing it to them with proper software directories, release logs, and documentations are then finalized as a process. The software when published to its community is then implemented, tested, deployed, used and reviewed by its community. The maintenance and support are carried out by the community. Due to these differences, open source itself holds a perspective as a development method.

2.2. Software Quality Assurance

Software Development Life Cycle (SDLC) is a recursive process. This process consists of different phases which include analysis, design, implementation and testing. The whole development cycle is paddled by Quality Assurance (QA). QA is a process by which one can assure that the software is quality software. QA plays a vital role in the development process. It is also important and requires proper addressing because customers are more concerned towards quality then quantity.

There are several procedures for assuring quality of a product or software. Figure 2.4 below shows a basic quality assurance process.

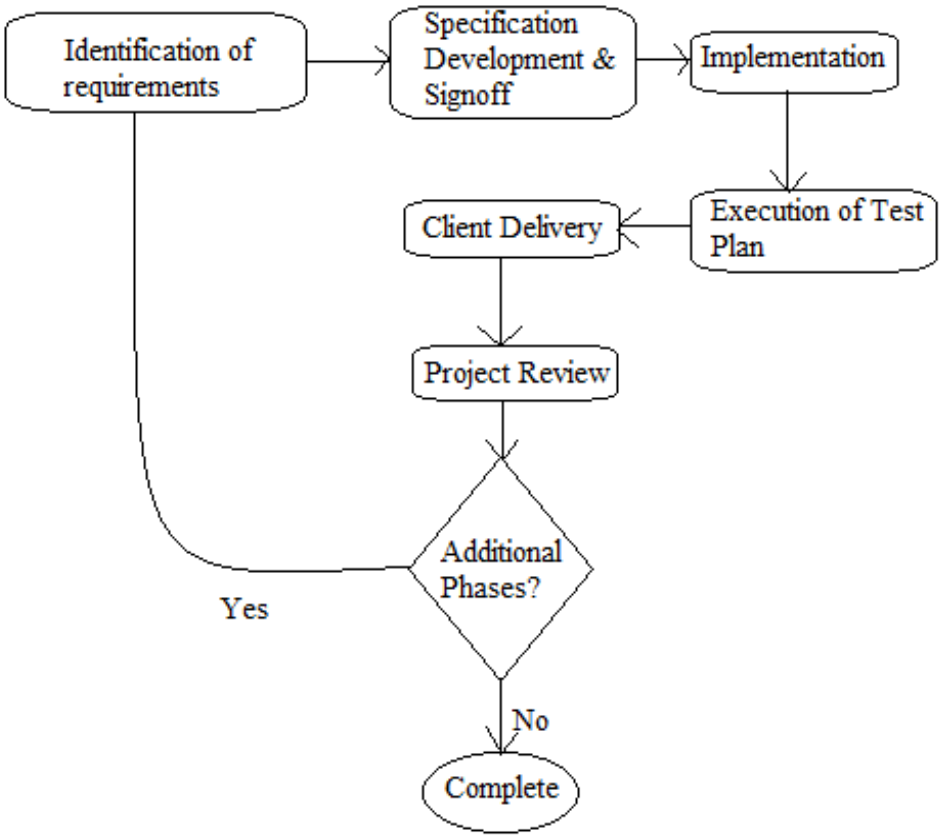


Figure 2.4 Quality Assurance process [36]

As seen figure 2.4 above, the QA process starts with requirements identification followed by development and implementation. The software is then tested and delivered. The final decision relies on review. Hence, one of the milestones in the typical QA process is review. The study shows that review is an effective mean to find bugs and flaws in any software. It has also been proven that reviews may be more efficient than testing. [32]

Review is a process in which the quality of work is technically evaluated. The evaluation is mostly based on the requirement set for the product and the end users. There are different types of review method. For example, in the context of software, the types are as follows:

- Code review,
- Pair programming,
- Inspection,
- Walkthrough, and
- Technical review

In addition to these types, in order to assure the quality, one needs to concentrate on the behavioral analysis of the software as well. Behavioral analysis is made by evaluating the quality attributes and by actually deploying and running the software. Few examples of these attributes are performance, reliability, usability, modularity and so on.

Review is the major factor in evaluating the quality of any software. However, quality is a vague term; researchers and scientists have their own definitions for quality of a product.

2.2.1. History of Quality Assurance

History of QA for software is not very long. The first QA model that was published for the software product was in 1976 by McCall namely McCall's Quality Assurance Model [11]. In later years this model was extended, redefined and merged to form some new models, frameworks and standards. The modifications were made based on products nature and requirements. These models are mostly used to evaluate products characteristics, therefore, often these models are categorized as product-oriented models. [6]

The models and frameworks presented in Table 2.1 below are considered as the primitive product-oriented models for software quality assurance. As we know our purpose is to provide a product-oriented review framework. Therefore these models will be the key part for this thesis work.

Table 2.1 Existing Quality Assurance Models

Quality Attributes	McCall, 1976/77	Boehm, 1978	FURPS, FURPS+ 1987, 1992	ISO/IEC 9126, 1991	Garvin, 1988
Aesthetics			Usability		*
Clarity		*Understandability			
Compatibility			Supportability		
Conformance				Portability	*
Correctness	*	*		Maintainability	
Device Efficiency	*	*	Performance	*	
Documentation		*	Usability		
Durability					*
Economy		*			
Features					*
Flexibility	*	*			
Functionality			*	*	
Generality		*			
Integrity	*	*			
Interoperability	*			Functionality	
Maintainability	*	*	Supportability	*	
Modifiability		* Maintainability		Maintainability	
Modularity		*			
Perceived Quality					*
Performance			*		*
Portability	*	*		*	
Reliability	*	*	*	*	*
Resilience		* Flexibility			
Reusability	*	*			
Security				Functionality	
Serviceability			Supportability		*
Supportability			*		
Testability	*	*Maintainability	Supportability	Maintainability	
Understandability		* Maintainability		Usability	
Usability	*	*	*	*	
Validity		*		Maintainability	

2.2.2. McCall's Model

In 1976/77, Jim McCall proposed a model for software quality which was initially used for space, military and in public areas. [9] The main aim for this model was to improve and assure quality for the software products.

This model contained a large volume of 55 quality characteristics (factors) which was later reduced to 11 (quality attributes) for the sake of simplicity. [8] One of the strongest parts of this model was the presence of interrelationships between these quality characteristics. McCall believed that, if the degree of detail for these attributes is high enough then, the quality of any product could be assured. [3] However, this model could not be considered as a complete solution due to its lack towards functionality measure of the software [7] and due to different types of software developed since then.

The software developed in 70's used to contain huge amount of code based errors, quality attributes in McCall's model were defined generally for the reviewing code level flaws. Furthermore, McCall's model was a step behind towards the measure of hardware characteristics. [7] In addition to this, the attributes like completeness and self-documentation are less meaningful in the earlier stage of software development and these attributes among others are indirect measures hence, according to Coté [11] and Pressman [12] McCall's model is not generic but slanted.

In 21st century, McCall's model is not a complete solution due to several reasons but back in 80's this model suited well for the type of software available and hence there were quite a few followers who adopted this model. For example, models like Murine & Carpenter's (1984) and Azuma (1987) are derived versions of McCall's model.

Out of 11 quality attributes present in this model the definition for many of them are still useful and hence could be used in certain extent. Figure 2.5 shows the partial McCall's model. In the context of OSS, attributes such as reliability, efficiency, usability, maintainability, portability and modularity are of greater concern. Therefore, we, in our LCM framework, included these attributes.

As it can be seen, the attributes such as Reliability, Usability, Maintainability and Portability have a set of sub-attributes as their measures. These attributes when are compared to that of other QA models differs in terms of these measures. This means that even if the quality attributes are adopted or chosen the measures are however changed and reformed in most of the cases.

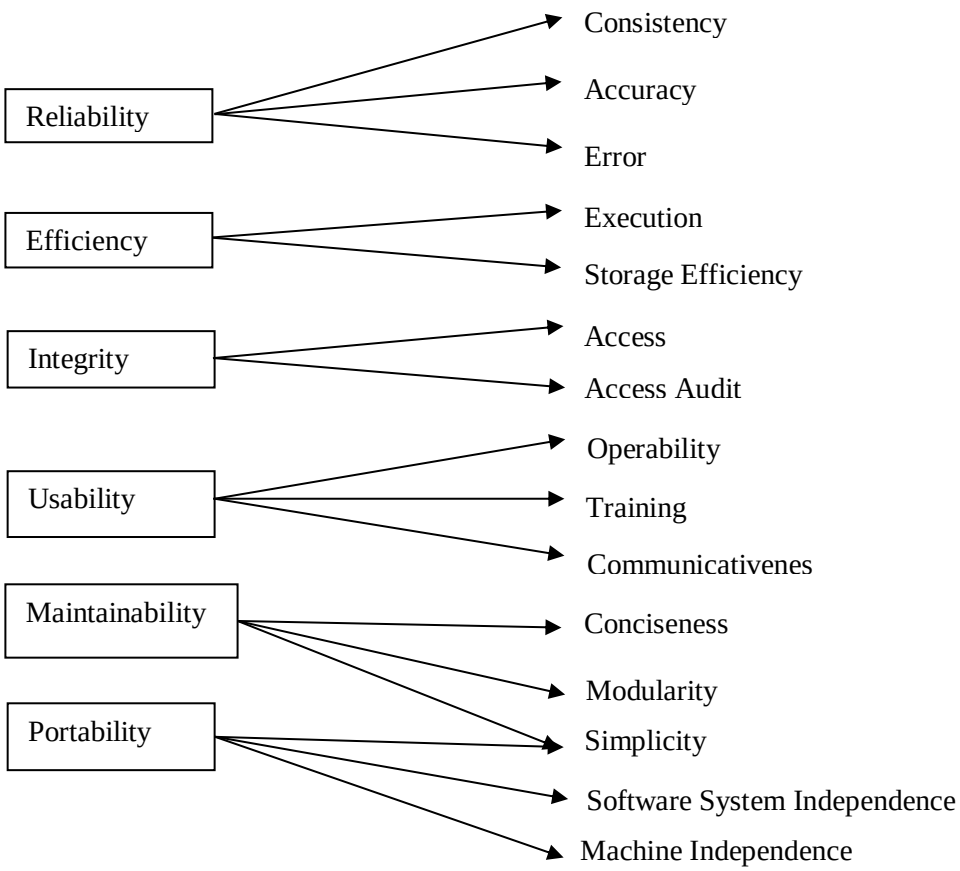


Figure 2.5 McCall's Quality Assurance Model [10]

2.2.3. Boehm's Model

A year later, in 1978, an American software engineer, Barry W. Boehm proposed another quality assurance model. Alike McCall's, Boehm's model also contained a set of quality attributes and their measuring factors.

In the existing McCall's model, Boehm added 8 new attributes namely clarity, modifiability, documentation, resilience, understandability, generality, economy and validity, which he found were missing. Boehm kept the ones that were relevant. He erased interoperability and testability which he found were less important for his model. This made his model more precise. [4] As discussed earlier, McCall's model was lacking the hardware measure which Boehm manages to overcome. However, the feature and functionality were still missing.

Boehm's model was leaned towards measuring the general utility of software through reliability, efficiency and human engineering i.e. integrity and communicativeness. He listed maintainability and portability as high-level characteristics. [7] For his model, maintainability was a prime issue. [13] Later in 1983 he proposed a specific model for the maintenance process known as Boehm's Maintenance Model.

Boehm's model could also be taken as a hierarchical approach where there are two levels of characteristics. The top levels which are Maintainability, Portability and As-is-utility concerned more to the end users, whereas the measuring characteristics (factors) or the bottom part is inclined towards the developers or technical personnel. [11]

Alike McCall's model, Boehm's model is tangled more on the bottom i.e. one factor helps in measuring at least one or more quality attribute. More precisely, in Boehm's model, a single factor Accessibility is used to measure both Efficiency as well as Human Engineering which makes this model less cohesive and less efficient to specify quality requirements.

The presence of accessibility in Boehm's model is redefined in our framework. The accessibility here is the measure of efficiency and human engineering whereas in our framework we include accessibility more than a measure. The need of open source requires us to include accessibility as an attribute with measures like availability and documentation.

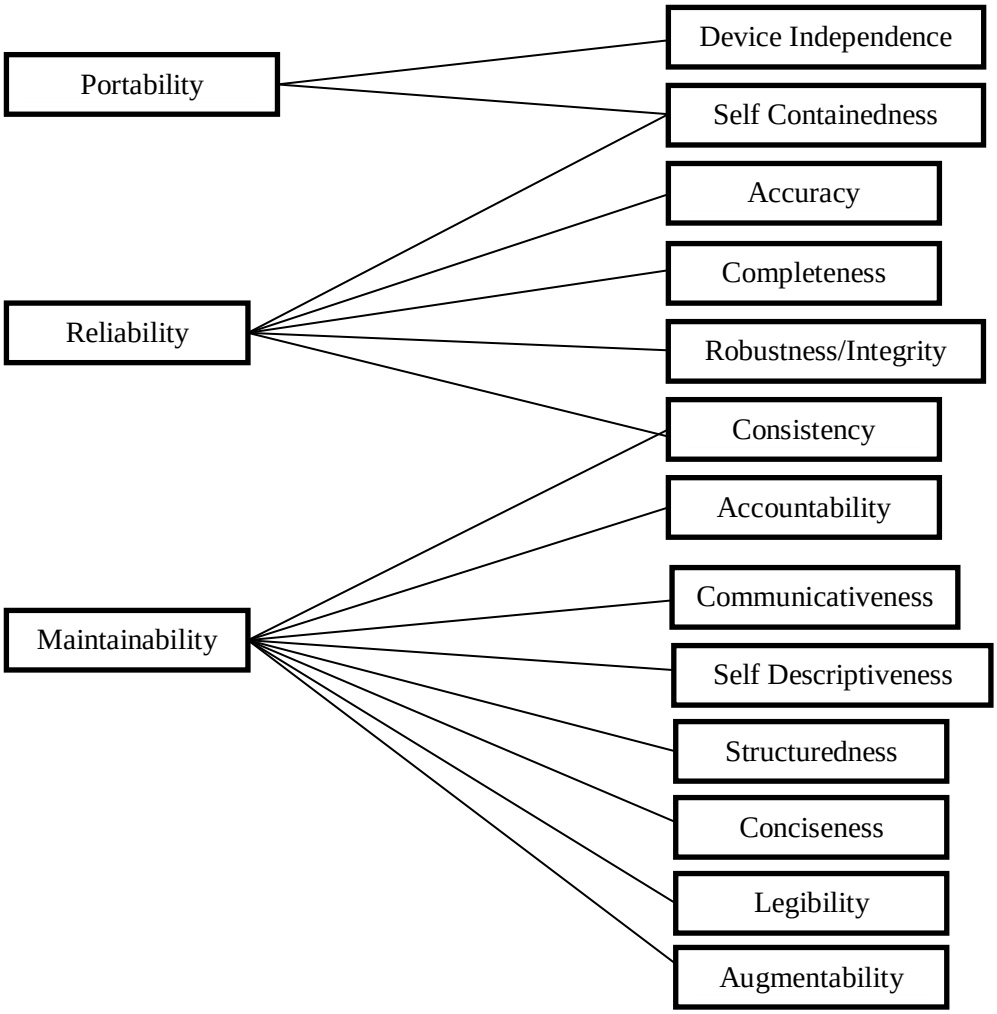


Figure 2.6 Boehm's Quality Assurance Model [13]

2.2.4. FURPS Framework

In addition, FURPS framework which was introduced in 1987 by Robert Grady follows similar hierarchy as that of the previous two models. This model is decomposed in such a way that one of the important leftover from the initial two models was put as a major category i.e. Functionality. The very basic structure for this model is shown in Figure 2.7 below.

This framework contains Supportability as a new attribute and reformulates Usability, Reliability, and Performance in a wider range. However, this framework lacks in measuring portability of software. The Localizability (Internationalization) as a measure of Supportability attribute is well put in this model which could be considered as a useful criterion to measure for software with localization as a quality requirement.

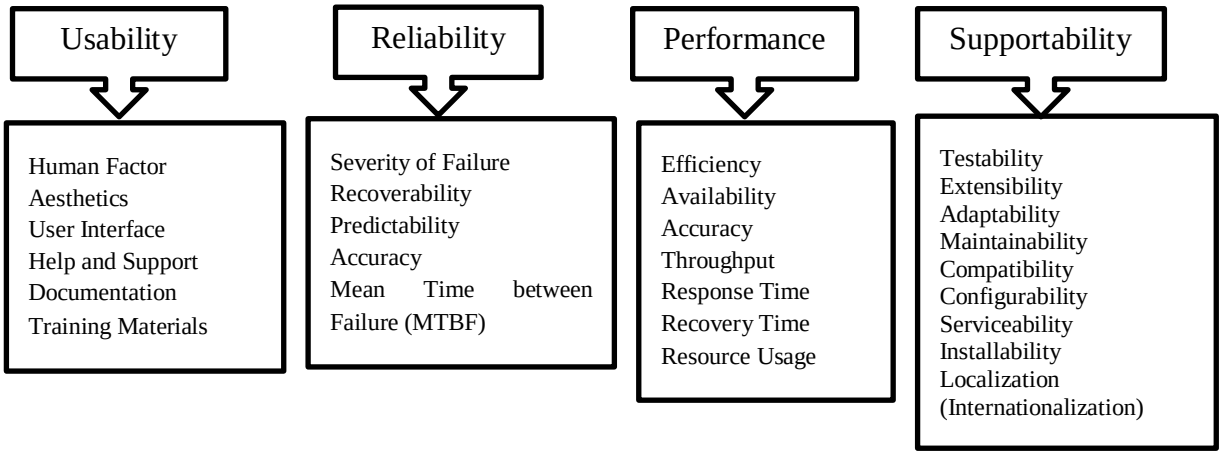


Figure 2.7 FURPS Framework

Later in year 1992, together with Hewlett-Packard Co., Robert Grady updated existing FURPS to FURPS+ framework, where + was the additional do's and do not's for implementation, interface and physical requirements [13].

2.2.5. ISO/IEC 9126 Standard

In 1991, based on McCall's and Boehm's quality models, ISO released a quality model ISO 9126 (aka Software Product Evaluation: Quality Characteristics and Guidelines). This model was comprised of 4 different parts

Part 1: Quality Model (2001)
Part 2: External Metrics (2003)
Part 3: Internal Metrics (2003)
Part 4: Quality in use metrics (2004)

This model contained Portability measure which was left behind in FURPS framework and also contained Functionality measure left behind in McCall's and Boehm's models making it a complete solution. All-and-all this Standard proposed 6 different quality attributes and their measures through multiple factors and sub-factors. Figure 2.8 below shows the contents of ISO 9126 standard. External metrics are the scale and method for

measuring software quality from the users' point of interest whereas internal metrics are the measures from the technical point.

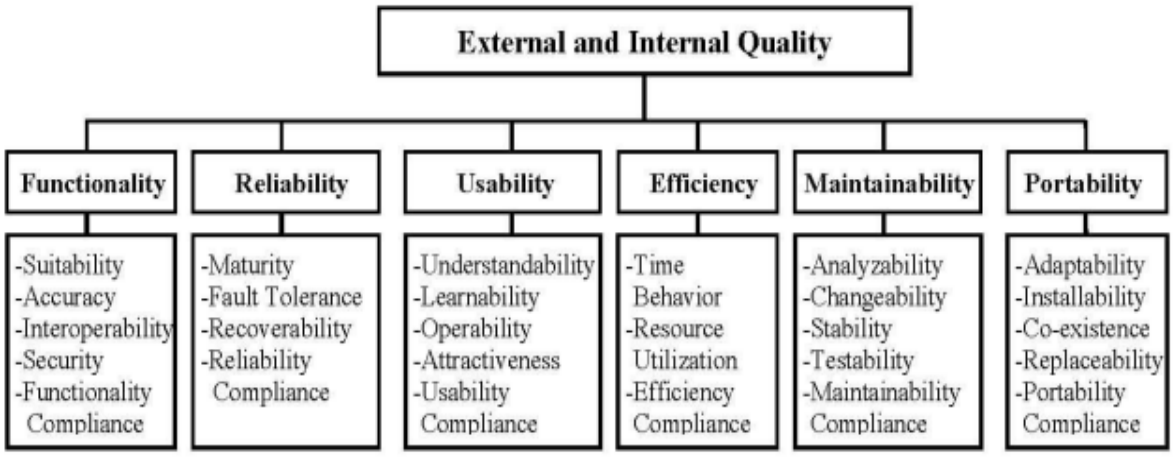


Figure 2.8 ISO/IEC 9126 Quality Model

It is to be mentioned that considerable amount of argument has been made [15] about which quality model is most useful based on the internationalization and coverage and ISO 9126 and its updated versions (part 1-4) has been chosen as the best due to the reason that ISO is built based on the international consensus and approval from ISO member countries. [15] However, for the context of OSS this model could not be a complete solution because it is lacking important measures like accessibility factor

Apart from these models, there are several other quality models proposed and published for different products and quality requirements. Some of which are fixed models and some of which are flexible ones. The fixed ones (including the ones defined above) provide a fixed solution for a generic software product through fixed set of quality attributes and its measures (factors and sub-factors) whereas in flexible models one is free to choose the quality attributes and their measures according to a specific quality requirement for that product. In flexible models the quality attributes could be indirectly defined as per required. Few examples of fixed models are as follows:

- IEEE Standards for Software Review and Software Quality Assurance Plans,
- Capability Maturity Model(s),
- Dromey (1995),
- Six Sigma,

One approach towards flexible modeling is Prometheus approach published in 2003. [14]

Due to the presence of many models and quality attributes, the review process has become more and more complex. Choosing the best and complete model is a bigger problem. In this thesis we extract a set of quality attributes from existing ones and present

them as a suitable framework for reviewing OSOS. These attributes are chosen based on the requirements for specific software hence this model will be a flexible review model.

2.3. Open Source Oriented Software

There are different types of software development approach. One of them is OSS. It is possible to view OSS in at least three different perspectives. In earlier sections we discuss about these perspectives which include community, licensing and method. OSS is therefore a different development method. In a typical OSS development there are few obligatory issues that must be addressed and verified. It is also possible to develop software as open source but with fewer variations in the setting. For example, software which complies with the definition set by OSI and FSF but the development starts and continues as in-house project by a small group of core developers. These core developers are solely responsible for designing the software, choosing the development settings, choosing the licenses, implementing, doing the market research, testing the software and finally releasing the software. The registration is to be made on public forge, and the software is to be release as OSS.

The software is, at the end, publicized to the open source community. The initial development does not include anyone else than the core developers. These core developers are not geographically diverse. These core developers or the project team uniquely owns the right for the initial state of the software, conflicting typical trend of OSSD approach. Since this development setting is possible and varies from typical OSS, we choose to call this Open Source Oriented Software (OSOS).

OSOS is a newly introduced term which suits as both, the type of a software or a development setting. Modified way of working than that of typical OSSD made it a different development approach. Figure 2.9 below shows a typical OSOS development approach.

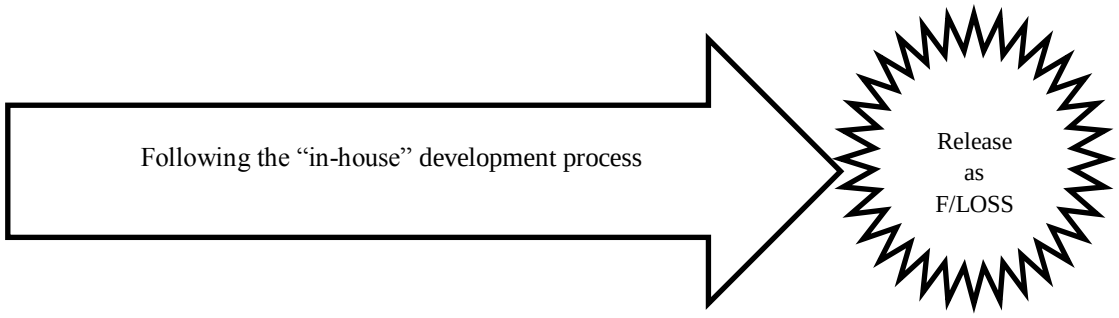


Figure 2.9 Typical OSOS development approach

2.4. Quality accessing tools

In addition to the existing software quality assurance models another thing to consider is the tools that has to be used for tracking different static results for the quality analysis. One of most important thing for the open source software development is communication. Since the developers, users and contributors are mostly geographically distributed; accessible and acceptable communications channel has to be used. One of the widely used channels for communication is mailing lists, which is indeed a tool. There could be separate mailing lists for core developers, users and contributors and other subscribers.

Another important role played by the tools in the open source development is communication but for a specific sector i.e. to keep track about the raised issues and bugs report for which bug/issue tracking tool is required. Furthermore, to check and evaluate the code quality for software, one needs to use the quality measurement tool which provides the static result for the codes and also the behavior result of testing.

Few examples of these tools that have been used in the recent development of open source software are Bugzilla [34], Mantis [35], Trac etc. Bugzilla is a bug tacking system or tool that helps developers keep track about the bugs in their program. This tool was initially used by Mozilla products. [34]. Similarly, Mantis is a web-based bug tracking open source software released under GNU GPL. [35]. Trac is a web-based project management bug tracking tool inspired by CVSTrac. Similar bug tracking tools are Redmine, EventNum, Fossil, The Bug Genie and WebIssues. Sonar [29] and its integration Bamboo are the overall quality management tool that keep track, analyze and measure the source code quality in terms of Response for classes, cohesion, code coverage and so on.

3. The LCM Quality Assurance Framework

As mentioned in earlier sections, in our context, the final product is being released as F/LOSS. Due to several aforementioned variations in the development settings, OSOS lacks a complete review framework. The development setting differs from a typical trend of OSS development. There are several quality assurance frameworks which could be easily adopted for reviewing OSS that follows the traditional settings of development. But the variation for OSOS unfortunately did not allow the available solutions to be adopted as a whole. Therefore for evaluating the quality of OSOS, we propose a Quality Assurance framework (a review framework) namely Licensing, Community and Method framework, in short the LCM framework. The name we choose is due to the reason that the perspectives to see OSOS mostly are closely related towards License, Community and development Method.

Our purpose is to provide a metric oriented review framework for reviewing OSOS. Therefore we chose 5 mostly used and accepted metric oriented/ product specific QA models as a basis of our LCM framework. Each of those models comprised of several quality factors and sub-factors. Most of these quality factors and sub factors are reused or redefined in all of those 5 QA models. Therefore we found it more appropriate to analyze individual attributes instead of the model as a whole. The LCM model will be interpreted on the degree of compliance towards these major perspectives including community, licensing and method, as that of OSS.

As mentioned above, there are certainly quite a many review models, frameworks and standards available for the software review. Most of these models are detailed code level and some of which are product specific. Since, the development settings we concentrate on is fairly new and different, none of the available models exactly fits to our context. We evaluated the requirement of our software from the chosen perspectives and came up with 17 relevant issues that must be measured. The in-depth study was done for all 17 attributes individually. Study showed primitive and mostly used models such as McCall's model, Boehm's model, ISO 9126 standard and FURPS framework have used some of these attributes with proper definition. With few or none alteration we adopted those measures in our framework. Table 3.1 below shows the extracted quality attributes from Table 2.1. This extraction is made on the basis of their availability in different QA models. In addition, Table 3.1 contains additional attributes including Accessibility, Modularity, Development Infrastructure, Product Registration and Software Marketing which were lacking in primitive models, but found to be relevant ones from OSOS point of view.

The available quality attributes and missing ones are categorized under three sections. If the attribute is influenced more by community requirements then that attribute is categorized as community compliance attribute. If any attribute complies more towards method, then so is categorized as method compliance attribute. Similarly, the licensing compliance attributes are chosen.

Table 3.1 Extraction of Quality Attributes

Quality attributes	QA Models	Remarks
Efficiency/ Performance	Present in all hence adopt	Adopt as that of FURPS
Maintainability	Present in all hence adopt	Adopt
Reliability	Present in all hence adopt	
Serviceability/ Documentation	FURPS and Garvin,	Adopt with reformation
Functionality/ Features	FURPS, Garvin, ISO	Adopt with reformation
Security	McCall have it as Integrity, ISO have it as one of the Functionality measure.	Adopt with redefinition
Portability	McCall, Boehm, ISO	Adopt
Usability	McCall, Boehm, ISO, FURPS	Adopt
Reusability	McCall, Boehm	Adopt
Licensing Compatibility	FURPS have compatibility as a measure	Adopt with redefinition
Modularity	ISO	Adopt
Conformance	Garvin	Adopt
Accessibility	None	Missing hence add
Development Infrastructure	None	Missing hence add
Product Registration	None	Missing hence add
Software Marketing	None	Missing hence add

3.1. Community Compliance

Open Source Software relies on its community. Success or failure of any open source software hugely depends on the effective framework made around the community and its requirements. All the major decisions taken for the development of the OSS must comply on the requirements of the community. As mentioned in section 2.1.1, Open Source as a community is an important perspective that could be followed. Hence, it is mandatory to analyze with deeper insight, the available quality attributes from the community point of view. In addition, quality attributes must comply with the community also for the reason that quality attributes makes the quality assurance.

There are guidelines on how to build a community, all of which, without missing, mentions about improving credibility, improving quality and developing ecosystem of support. This mostly holds true for the developers' community. Other than that, if we are talking about the users' community then more importantly, user friendliness, ease of support and accessibility tops the list.

Following are the quality attributes which are available in the previous models and comply with the open source community:

- Reliability
- Maintainability
- Efficiency/ Performance
- Serviceability/ Documentation
- Portability
- Usability
- Conformance

These attributes for the quality assurance act in accordance to the community requirements.

Reliability

In Table 2.1, it is shown that Reliability factor for quality is being chosen by all of the QA models Product operation factor in McCall's model, As-is Utility in Boehm's model, External metric in ISO 9126 and Non-Functional attribute in FURPS. And accuracy is chosen as the measure for this attribute.

In our context, when the software is OSOS, this attribute should be placed as community compliance attribute due to the reason that the dependability on any OSOS provides higher credibility to its developers at first place, and it is also a factor of motivation to work on the software. However, the measures for this attribute remain same (Figure 3.1 below) as that of ISO 9126, McCall and Boehm i.e. Recoverability, Fault Tolerance and Accuracy. But in oppose to McCall and Boehm's model, the Completeness is not considered as a measure for Reliability because we agree that "Release early, release often" mantra by Eric Raymond in his book The Cathedral and the Bazaar best suits for the open source context.

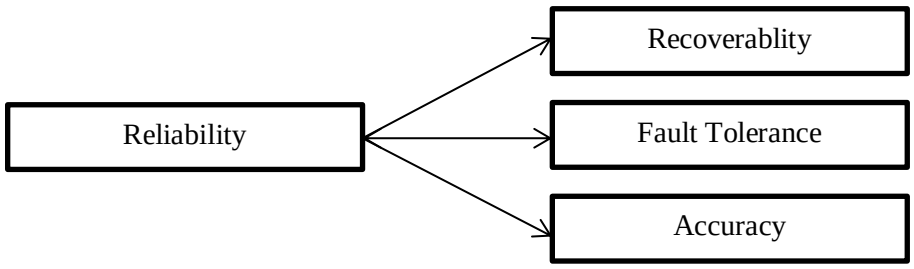


Figure 3.1 Reliability and its measures

Maintainability

Maintainability is an ability of any software to bear specified change in itself. In other words it is the ability of any software on how easily it could be modified. Maintainability index of any software is affected by the quality of the source code. Alike OSS, the architecture of the OSOS is likely to change every now and then because there are always new requirements and ideas put forward by the community members. Therefore, if the initial source code is rough and scattered i.e. does not follow a predefined pattern then maintaining the software or changing it according to the changing requirements would be problematic and hence will affect the quality.

As we can see in Table 2.1 McCall, Boehm and ISO 9126 have Maintainability as a separate quality factor, whereas FURPS on the other hand have counted it as a measure of Supportability. In the context of OSS, Maintainability is a vital requirement and is definitely affected by the community in a higher degree. Hence it should be placed as a different quality attribute with following measures:

- Structuredness
- Simplicity
- Consistency
- Self-descriptiveness
- Testability

In our context, we follow and accept McCall's interpretation of Maintainability measures and hence choose Simplicity, Structuredness and Self-descriptiveness. In addition to these, we choose Testability as a Maintainability measure from Boehm's and ISO model. Consistency on the other hand is chosen from Boehm's model because stable and steady software yields Maintainability.

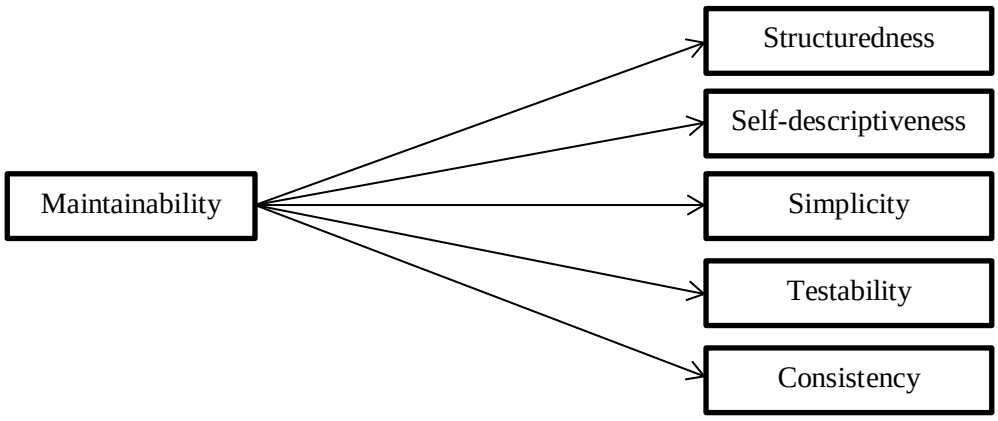


Figure 3.2 Maintainability and its measures

Performance/ Efficiency

Time is valuable. Perhaps this is the reason why users and developers choose not to wait and waste their time on a mere application. One of the reason which affects performance is hardware. Therefore these users and developers buy systems which have higher configuration and are expensive. These users expects that the software which are developed to be run in these high configuration systems are efficient enough and the design decision made on these software for better performance (in regards to response time, throughput and resource usage) are correct and valid. Hence Performance and/or Efficiency factor is adequately important quality attribute that must be reviewed for securing quality for any software. When the software is Open Source, like in our context, especial attention needs to be given in reviewing performance because in most of the OSS these user expectations are more, and not to forget OSSs are built around communities which contain users and developers who decide on the software quality.

As in Table 3.1, it could be seen that (as called) Efficiency is included in all the QA models reviewed except that FURPS finds it suitable to call it Performance instead. In our context we choose to call it Performance/ Efficiency and include following sub-factors as its measure.

- Time Behavior
- Resource Utilization
- Validity

Here Time Behavior is chosen, considering the fact that it consist measures like response time and throughput. Whereas Resource Utilization is measured by resource usage by the system and Validity gives the accuracy of the result.

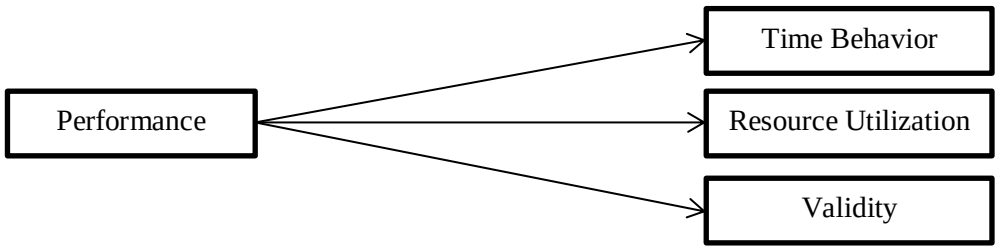


Figure 3.3 Performance and its measures

Serviceability / Documentation

Serviceability, in general, is an attribute which concerns about the services, help and technical support for the software. In the context of OSS, this non-behavioral requirement is a design decision made in order to achieve software ability on supporting, monitoring, identifying and solving the raised issues by the concerned community members. These issues can be related to installation, deployment, exceptions, faults, errors or debugging. Mostly these serviceability criteria for OSS are measured via Help desk support, network monitoring, event logging, and documentation. Documentation here refers to both the technical as well as non-technical documents that are related to the software. For example, Software Architecture Document, Specification Requirement Document, User Manual, Data Dictionary and so on. This attribute is chosen in accordance to community because all the services that are provided by the software are for its users and developers who form the open source community. In addition, this attribute directly relates to the Maintainability of the software.

As seen in Table 3.1 FURPS and Garvin's model have Serviceability as a quality attribute. However these models have used this attribute as a measure to Supportability. We prefer to choose Serviceability instead of Supportability because providing support to software is a part of overall service.

Following are the measures of Serviceability:

- Documentation
- Supportability
- Help desk
- Fault/ Error Tracking
- Localization (Internationalization)

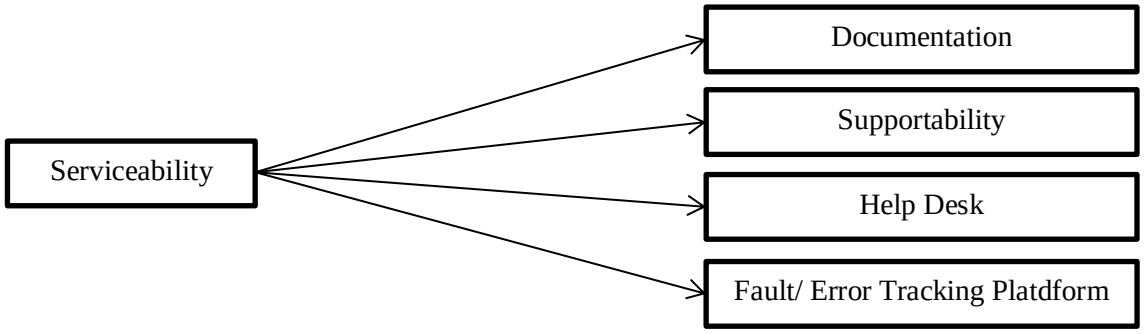


Figure 3.4 Serviceability and its measures

Portability

Flexibility in software is an important concern that needs to be addressed during the design phase of SDLC. Flexibility on the other hand is a portability measure which generally means the ability of software to adopt changes in different environment. In current day scenario there are different computing platforms, for example, Microsoft Windows Operating System, Linux based Operating systems, Mac OS X and so on. The users are free to choose any of these platforms for their computing. If the software is not portable while changing the platform then there is certainly increase in the development cost and relative decrease in the number of users and developers which affects the community. According to ISO 9126, the software could be made portable by adopting Object Oriented design and implementation. As we can see in Table 2.1, except FURPS all the other QA models found Portability as an important quality factor. However the sub-factors used for measuring this factor varies from model to model. In our context we choose following measures, which we found are in accordance with the open source community.

- Platform Independence
- Adaptability

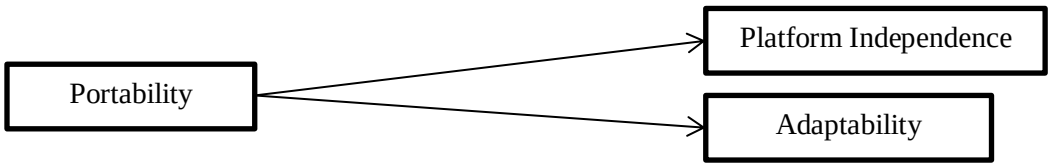


Figure 3.5 Portability and its measures

Usability

Usability is one of the most important characteristic of the software QA. If any software, irrespective of its type and nature of development, is complex in term of using then the users of these softwares are definitely limited. Usability assurance on the other hand is one of the key holes to achieve quality assurance. [18] Usability helps in increasing the users and their productivity, which in the context of open source is vital. Productivity in the sense that users help in tracking down the errors, defects, coming up with some innovative ideas for further development and so on. In addition to these, the operational risks as well as costs could be reduced if there are more users involved actively or passively in the development and use. In a nutshell, usability is an ease to use. It helps to increase users, track down the errors and fix them which acts in accordance to the community. Current research and practice in Usability and quality assurance shows that users are the main source of reporting bugs and are likely to be the co-developers therefore it is recommended that users must have a proper and adequate understanding about the practices and context of use. [18] In the context of OSS and community, usability must be addressed

with higher importance for the reason that critics are emphasizing on the fact that usability is almost absent (or present as low priority requirement) in OSS products and also OSS is mostly designed for and by the users. [19]

This measure of QA is present is all the QA models except Garvin. However, Garvin without failing mentions aesthetics and perceived quality which covers usability requirement. We have chosen following measures for the usability of OSOS:

- Understandability
- User Interface/ Attractiveness
- Operability

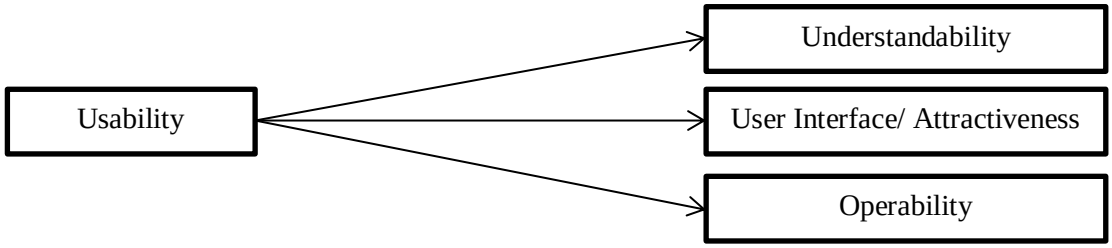


Figure 3.6 Usability and its measures

Conformance

According to Garvin, one should not rely on a single set of definition which is likely to cause problem. This is why we can have our own set of definition for different terms provided that the new definition must not conflict the original meaning.

Conformance is a matter of matching between the product design to the internal and external standards set for the product. [26] In this definition, Garvin has not explained, what are internal and external elements? This therefore, in our context could be the organizations. Internal organization is the one that is developing the product whereas the external organizations are the ones for whom the product is being developed and other third parties related to it. In other words they are the organizations that show interest in the product either for use or for further development. The role of outside organization or the external elements in OSS directly correlates with community sustainability and governance [33].

For example, if company X is developing an OSS primarily focusing for company Y then there are set of requirements from Y that must be met by this software. Also if the software is meant to be released as OSS then company X must take care of the requirements set by OSI and/or FSF, making OSI, FSF, and Y as external elements and company X itself being as internal element.

While performing a quality review of an OSOS one must take care and review all the compatibility documents/ requirements from internal as well as external organizations.

In addition to above attributes which were adopted from the 5 QA models we have following quality attributes which were missing but are relevant in our context.

- Security
- Modularity
- Accessibility
- Software Marketing

Security

According to ISO 9126, the software security is its ability to protect and prevent its information and data from unauthorized access and at the same time the software must not restricts the authorized ones to access the data and information available in the system. [7].

It has been defined by Firesmith [25] that due to the property possess by security in preventing the malicious harm security is a dependability factor for the software users and hence it is a quality factor. The major aspects which a secured application should contain are in communication channels (internal and external connections) and data channels. [25]

Software critics and developers often claim that security in the Open Source development environment is generally ignored and is easy to invade the system due to the reason that the source code and all the product information is public and is made easily accessible to everyone. In contrast, we would argue that open source software are not always a complete solution for first couple of releases, they are the prototypes and something to work on [28]. Also OSS are freely taken and molded according to ones need. Hence, in the later releases security are important and should be taken as a customization point.

This factor, in our context is chosen as a community compliance attribute because the developers are users and users can be developers which is why the common concepts underlying security is best known and analyzed by the community members.

As we can see in Table 2.1 security is missing in most of the reviewed quality models. However ISO 9126 have it as a functionality measure. On the other hand McCall's and Boehm's model contains integrity instead. In our context, we found that Security is vitally important concern to be addressed and without which the quality review for any software is not possible. Hence we propose Security as a quality attribute and integrity as its measure.

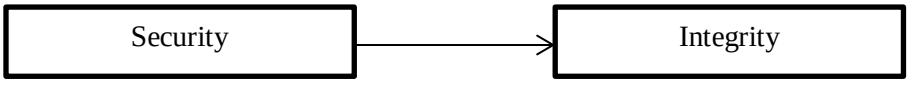


Figure 3.7 Security and its measure

Modularity

As it is seen from Table 3.1, Modularity has not been included as a separate quality factor in any of the compared models. However we found Modularity as an important attribute for assuring software quality.

Modular architecture of the system has a firm grasp on complex issues related to design and production. There have been comparative studies [20] and arguments [21] on how modularity helps in producing quality software in terms of redesigning and maintenance.

In our context when the software is Open Source and the focus is to be made on the community requirement, I would argue that Modularity requires a lot more attention as a quality attribute because it is directly affecting another important attribute i.e. maintainability. OSS must be flexible for redesign and the artifacts must be made accessible to everyone, therefore with the modular architecture it is easy to track and separate the interdependencies between the packages and hence will result in less-effort redesigning. In addition to this, software customization, which is highly probable for OSS, will come in handy.

Furthermore, by following a particular trend of modular programming, previously developed source codes could be reused with very few or even no change, which will definitely save developers time and effort resulting in a quality community software.

The measures for this attribute are number of tangled entity (for example packages) and dependencies between them.

Accessibility

Alike Modularity, another missing attribute from McCall's, Boehm's, ISO 9126, FURPS and Garvin's QA model is Accessibility. One reason on why these models failed to include one of the primary quality attribute in their model may be that these models were least concerned about the open source software. In our context, accessibility is a must quality attribute that has to be verified and reviewed even to mark the software "open source". As mentioned in earlier sections, the first and the foremost criteria to be open source product is to make all the source code and product documentations public, failing which the software could not be an F/OSS. This requirement is directly related to community in the sense that usually OSSs are designed for and developed by the community. All the operations related to development of open source software are handled by the community itself. Hence the option of accessibility, accessibility of source code, accessibility of technical and non-technical project documents and all required information must be given to the users and developers of the community.

This attribute could be measured through the availability of source code along with binary, executable and all the related product documentations. The documentation measure is common to both the Serviceability attribute as well as Accessibility.

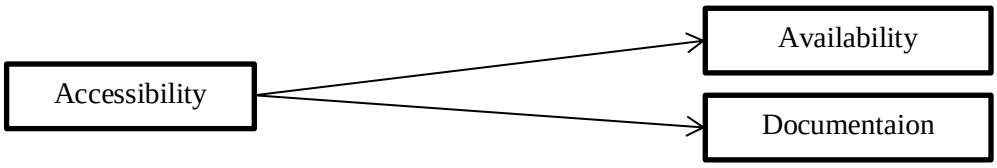


Figure 3.8 Accessibility and its measures

Software Marketing

Marketing, either for commercial products or open source software, is of equal importance. The strategy however may differ. Marketing is an art [27], art of selling products. The core marketing concepts must be understood with ease and has to be implemented as an everlasting process of product development. Setting up a target market segment, knowing the expectations from these segments, analyzing segment's need, want and demand followed by the product advertisement to these segments are some of the core marketing concepts. [27] Understanding these concepts is of equal importance from the initial phase (design) till the maintenance phase.

Software marketing is separately defined for both social and managerial perspective. Choosing the appropriate one, creating a strategy and implementing it is how one could improve the number of users, developers and achieve financial assistance. This attribute has been chosen as the community compliance component for the reason that, marketing makes software visible and it is vitally important for delivering quality product. As mentioned earlier, community beholds the control to the software. Therefore where and to whom software has to be publicized is equally important.

3.2. Licensing Compliance

Alike in OSS, Licensing in OSOS is an important characteristic which is to provide flexibility to freely exchange and use information among all its users and developers. It is the freedom to freely redistribute and modify the software is covered in all the available open sources licenses. There are thousands of open source licenses available in todays' market, all of which, without failing shares a common idea of redistribution and modification flexibility. [22]

Whenever OSS is to be developed, one must offer a prime concern towards its compliance on licensing, for the reason that Licensing of OSS makes it distinguishable from other types of software for example from proprietary or commercial software.

There are few important aspects that must be verified on or before releasing any software as Open Source. One of them is to choose an appropriate licensing scheme for the software. All the open source licenses include the basic requirement clauses such as allowing derivation and distribution of the original work. However, these flexibilities provided by different licenses might vary on the nature of the software, its intention and the circumstances, which is why choosing appropriate licensing scheme is important. After the

appropriate license is chosen for the software, all the components that are integrated with this software must meet the terms of the chosen license.

For example the GNU GPL license is more restrictive than the BSD license. GNU GPL allows to use, redistribute and change the software, but also requires the changed version to be licensed as GNU GPL whereas BSD being a permissive also allows to use, change and redistribute the software (even to proprietary one) but does not limits the modified version to be BSD. Therefore, if one wants their codes and documents and software itself to be more flexible in terms of redistribution then it is appropriate to use BSD instead of GPL. The point here is, while doing a quality review of an OSS this aspect of the development must be checked and verified based on the initial requirement of the software which is meant to be Open Source.

Another important thing to be assured while doing a quality review of OSS is to check and verify if all the assimilated software components put up with compliance to each other. For example, the licensing compatibility between Apache License version 2.0 and GPL version 3 is omnidirectional i.e. Software with Apache version 2 licensing scheme can be included in projects following GPL version 3 licensing scheme but the reverse does not hold compatible. [23] There are several such examples for the compatibility which must be taken into consideration without failure.

From the Table 2.1 it could be seen that quality attributes such as Reusability and Compatibility even though are present in the primitive models but the context that these attributes are used is slightly different from that of ours for the reason that we are reviewing OSS and these models were mostly concerned to commercial products and hardware lines.

Therefore we propose Compatibility and Reusability factor as the review factor for open source software which complies with open source licensing.

License Compatibility

In open source compatibility has at least two meanings; machine compatibility and license compatibility. Machine compatibility is the ability of any machine to work in or run together with another machine provided that they are connected with some medium. Whereas licensing compatibility is the ability of different software or its components to comply on different software licenses.

This attribute, even though has been used as a review factor in various models have a specific requirement in our context. The only important reason behind choosing this attribute is to measure the level of compatibility between different Open Source licenses and their use which is indeed an essential step in open source review.

Legality is dangerous, it is to be remembered that even different versions of same license may not necessarily be compatible to each other. It entirely depends on the clauses and conditions that are used in those versions. For example “GPL version 2 by itself is not compatible to GPL version 3” [24]. Therefore it gives an utmost essence to review and verify that all the software components, code artifacts (if are taken from different OSSs) are compatible to each other.

The measure for this attribute could be some support tools that can analyze libraries for binary, codes and even look for the licensing block on top of all the classes (if Object oriented approach is used) for compatibility. There are several such tools available in the web for example Open Source Compatibility Metrics (OSCoM), Code analyzer, Clirr, Sonar and so on.

Reusability

According to Firesmith, Reusability is a development oriented quality factor which gives simplicity of reusing the existing applications or components. [25] According to him, this quality attribute plays a primary importance prior and after the main show i.e. while developing and during maintaining but not equally during actual application usage by the users. Similarly McCall categorize this quality factor together with portability and interoperability and called it as one of the product transition factor. Boehm, ISO 9126, FURPS and Garvin on the other hand does not emphasize on this factor in their models.

While talking about this factor in the context of Open Source we would accept Firesmith's definition but with slight modification. Here the ease of reusing the application and components should also be extended deeper towards the code level. Similar to modularity, which covered modular architecture, design and code: reusability concerns to both reusable components as well as reusable codes.

The measure for this attribute is the count or the ratio of unique methods in a class (more the better) because, with modules containing more unique functions which, if separated, could make the separated code block possible to act individually in other programs. [7]

In addition to these attributes, Trademark and Copyright are two important issues that need to be verified and complied for the quality assurance of OSOS.

3.3. Method Compliance

As mentioned in section 3.3, open source could be viewed as a development method. Development method consists of several processes. These processes are directly dealt from the administrative level in most of the proprietary software. However, in the context of OSS/OSOS, these matters are shared and handled by project manager, core developers and even some other community members. Alike OSS, OSOS is also an open platform allowing all the people to freely communicate their views and opinions, as a result of which the best possible solution is chosen, hence, matters including legality, registration, marketing and infrastructures directly relates to their responsibilities.

Among the chosen QA models, we could not find the quality attributes which would directly or indirectly indicate the solution to verify and validate issues relating to registrations, infrastructure, marketing. Hence, in our model we propose Software Registration, Software marketing and Development Infrastructure as three major quality attributes that must be reviewed in order to achieve full quality of OSOS. In addition to this, Conformance is chosen from Garvin's model which is defined as an attribute which could be used to verify and validate the legality issues.

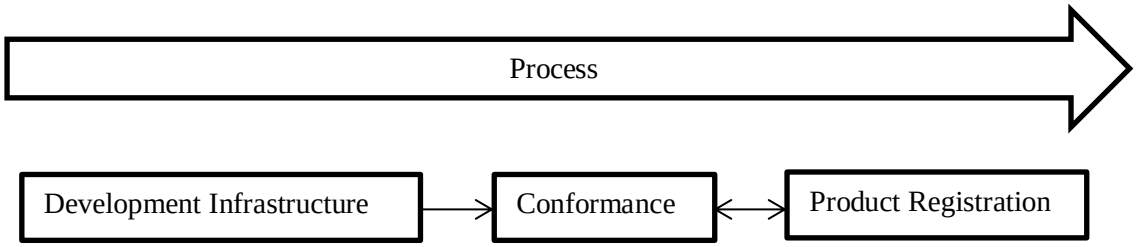


Figure 3.9 Development method flow

Product Registration

Product registration has at least two meanings. In the context of commercial products, registration means to put a product in the company help and support group for some of the following reasons:

- To track the numbers of people using a product,
- To be able to send important and available updates,
- To provide efficient support,
- To provide information on events and so on

In addition to these reasons, for other products, specifically for OSSs, registration is done also for following reasons:

- To make the whole package (including executable and binaries/source codes) available for download (via forges) to the interested users, for free. (accessibility)
- To track the number of downloads and views

Product registration or software registration, in our context is a method compliance component which comprises of basically two major issues namely software registration and license registration. These attributes must not be left unattended and should have a higher degree of importance in order to receive a quality tag for the OSS. It is obligatory that the registration either to its license or software overall, has to be made before the final product release. It is one of the initial milestones that must be checked as a startup factor for OSSD as well as OSOS development, which is why we purpose this attribute as an important review attribute for our LCM framework.

Development Infrastructure

Infrastructures in general are the most important components that are required to sustain; and development infrastructures or more precisely software development infrastructures are the crucially important components without which the software development would be difficult to imagine.

In the context of both OSSD and OSOS development, managing these infrastructures is very important. Some of the major infrastructure/ development components, apart from manpower, finance and hardware, are for example, Source Control or version control tools, Continuous Integration tools, staging tools. In today's software market there are thousands of such tools available for example Git for hosting or version control, Hudson, Bamboo, TeamCity, CruiseControl (.NET) for integration and Maven with both central and local repository for deploying software onto servers for testing purposes, prior to deploying them fully into production.

The main point here is, since there are lots of these open source tools available, choosing an appropriate one, which would be compatible with license and handy for the developers, is a tough job. Hence, one needs to check for the compatibility and ease of the used components or infrastructures before or during the implementation stage; most precisely before the release. A continuous research for making the right choice may be required throughout whole development cycle.

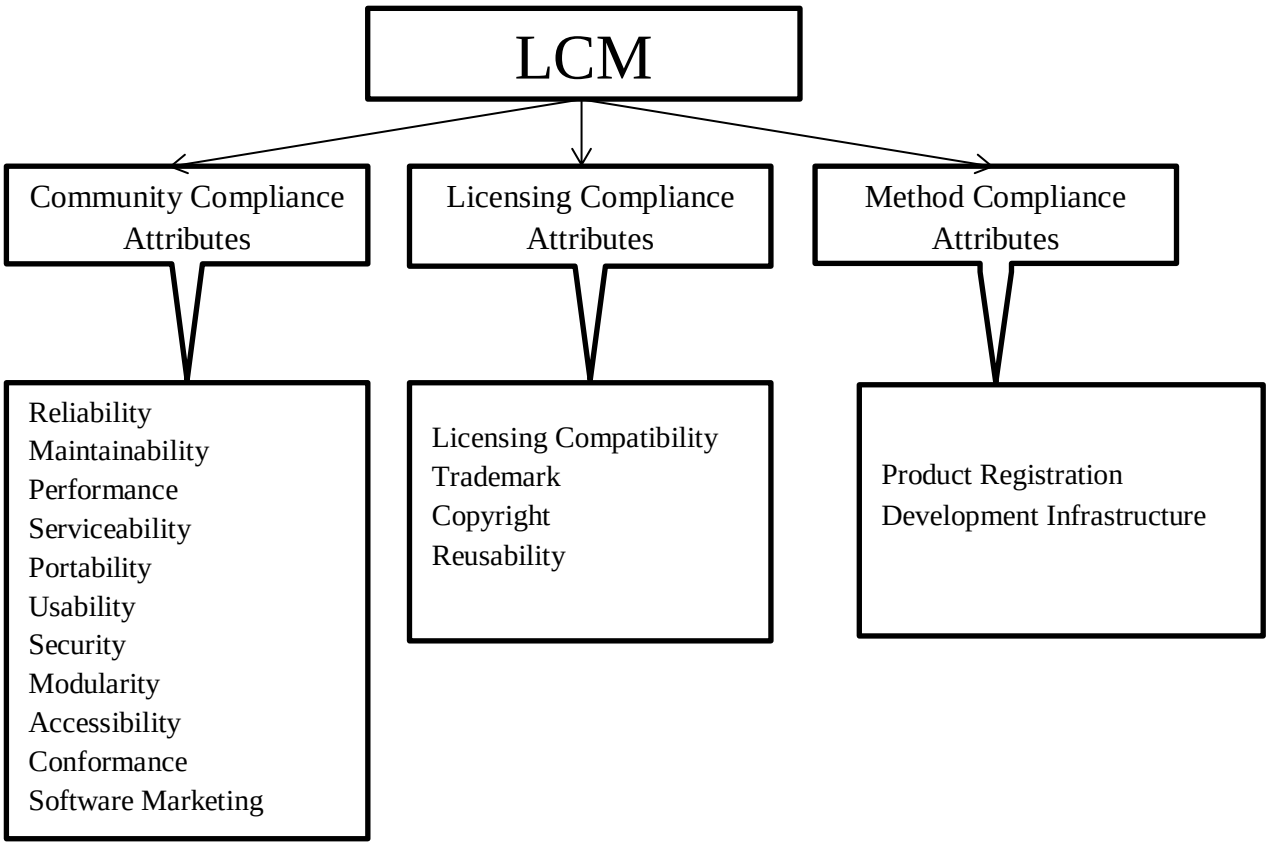


Figure 3.10 The LCM Framework

In the figure 3.10 above, we present the final result framework which insures that any OSOS, if reviewed following the proposed attributes and measures, will result as quality software. In the following chapter, we present the case study where the LCM framework was used.

4. Case Study – FLOSS SOLA

The LCM framework was used over an open case project called SOLA. In this chapter, we present the result. The categorization and interpretation of each achieved results are also discussed in this chapter. We also revisit our purpose in the discussion section of this chapter.

4.1. Solution for Open Land Administration (SOLA)

The LCM framework was used over SOLA. SOLA is developed by United Nation (UN) FAO as OSS. SOLA aimed to be used in developing countries initially for Nepal, Samoa and Ghana. FLOSS SOLA is OSS; however, the development setting used had several variations from a typical trend of open source software development. Even though SOLA software was meant to be Open Source, the development started and continued as in-house project by a small group of core developers concentrated in FAO Rome. These core developers were solely responsible for designing SOLA, choosing the development tools, choosing the appropriate license, implementing, doing the market research, testing, building a community and finally releasing it to the community. The registration was made on public forge and the first release was Free/Libre OSS (FLOSS) SOLA.

This project was taken as the case study for the conclusion made in this thesis work. Four different versions of SOLA application was reviewed and analyzed both statically and behaviorally. The marketing of this software, to the potential users and developers, were also made before concluding this thesis. Even though the conclusion was made solely based on this software application, we believe that the result of this thesis is useful and could be used as a tested framework for reviewing any other OSOS developed using similar development settings.

4.2. Results

In this section we present the results and experiences that were obtained while implementing our review framework to FLOSS SOLA. The results are divided in two wide categories namely static and behavioral sections for community attributes. The static review was carried out with the help of a code analyzing tool named Sonar which contains several metrics whereas behavioral analysis was made with the help of the results of static analysis and executing the application. All the attributes for dynamic analysis falls under the category of community compliance. The review was made on four different versions of SOLA application namely development snapshot (1st), Alpha release (2nd) and Customization release (3rd) and Release Candidate_v1.0 (4th). The review reports are accessible from SOLA homepage www.flossola.org.

As mentioned, the static analysis is the base for most of the behavioral results; the discussion on code quality is made first.

Static Analysis (Code Analysis)

Static analysis or code analysis is basically done to define and analyze the software quality objective by executing program built. The importance of code review is important due to the reason that most of the activities, especially in the open source development, happen at the code level [28]. There are several tools available for the code analysis. Here for the SOLA application the tool that was chosen was Sonar. This tool contains several metrics which helps in determining the code quality using internal calculations and universal mechanisms (testability, readability, bug tracking engines [29]). Metrics that have been used by Sonar application includes number of statements, complexity (cyclomatic), tangling index, responses to a class, connected components, violations (with severity levels), dependencies (files and packages), code coverage, architecture and design, duplication and unit tests (with Bamboo integration) [29]. These metrics available in this tool directly relates to the dynamic analysis or the behavioral aspect of the software quality. In Table 4.1 below we have the list of Sonar metrics and the results that were obtained for three different versions of SOLA application.

The importance of static analysis for review is mostly related to the community compliance attributes. As mentioned earlier, most of the metrics that are present in Sonar directly influence the behavioral aspect of quality. For example if the Response for Classes (RFC) value for a class is large, it means that, when the object is invoked for this class the number of methods that could be executed is more, which results in difficulty to understand, debug and test the software which is indeed a maintainability issue. Another example is Packet Tangle Index where the index gives the tangling level of the packages, the best value is 0% meaning no cycles. This index has to be reduced in order to get less tangled packages as a result of which we get more modular and reliable code helpful for the community to extend the software. Even though the results obtained by performing code review were not promising. It certainly helped in improving SOLA quality.

Each of these metrics were analyzed individually and based on the results obtained the suggestions and recommendations were made. The review was normally carried out before the release and the results and suggestions were used for the next release of SOLA. It could be seen from Table 4.1 that some of the metrics have low improvement due to the reason that in less than a year (Aug 2011 - April 2012) there were more than 50 thousands lines of code increased. Therefore maintaining the same state was harder than expected. In some sections for examples Rules and Violations, the numbers have increased from 3000 to 7000 due to increase in minor and informative violations. However, most of the critical and major violations were eliminated which was indeed an improvement. On the other hand, code coverage was a total disappointment. Even though it was repeatedly reminded no actions were taken for this metric. Therefore for the first version of SOLA we chose to find some object oriented anti patterns including God Object, boat anchor, catch-all (Figure 4.1), magic numbers (Figure 4.2) and circular dependency (Figure 4.3) all of which have intents and suggestions.

```
try {
    value = new BigDecimal (txtValue.getText () );
}
catch (Exception e) { }
```

Figure 4.1 Catch-all anti-pattern

According to Figure 4.1, the code snippet from sonar shows that the initial version of SOLA application consisted of object oriented anti-pattern known as catch-all. It is clear that the code is trying to catch an exception object 'e' which in itself is an error. It was hence recommended that errors as such should be avoided.

Similarly in Figure 4.2 below, it is seen that numbers such as 23, 59 are used out of nowhere. These constant numbers are treated as "magic numbers" which is an anti-pattern in Object oriented programming.

```
tabValidate.getColumnModel ().getColumn (3).setCellRenderer (ir);
calendar.set (Calendar.HOUR_OF_DAY, 23);
calendar.set (Calendar. SECOND, 59);
```

Figure 4.2 Magic Numbers anti-pattern

In addition to these, Figure 4.3 below shows yet another anti-pattern which was present in the architecture of SOLA application. This anti-pattern is namely circular dependency.

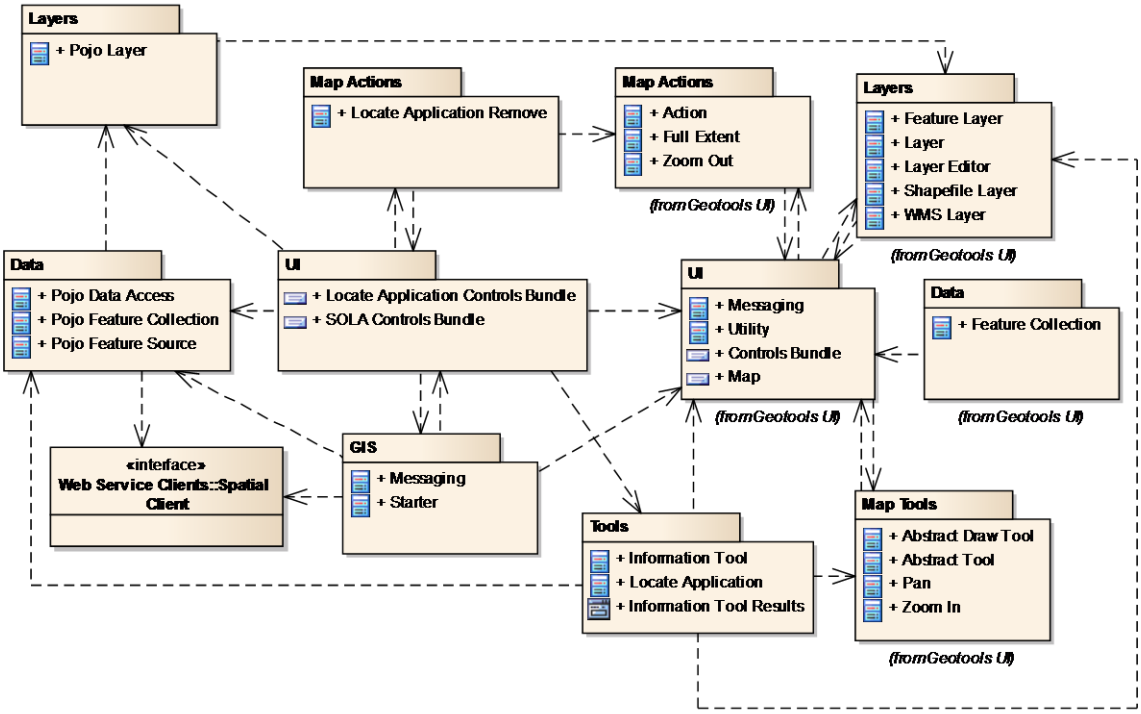


Figure 4.3 Circular dependency

Table 4.1 Comparison result for four different releases (Static)

Metrics	**1st Release**	**2nd Release**	**3rd Release**	**4th Release**
Release date	August 2011	December 2011	April 2012	September 2012
Lines of code (LOC)	29,999	64,611	84,004	Decreased 50,821
Comments	24,3%	22,8%	22,4%	Decreased 19.9%
Number of Classes	532	1.015	1,230	Decreased 768
Response for Classes (RFC)	12/class	15/class	16/class	14/class Improved
Rules and Violations	3.205	6.424	7.216	2.436 Improved
Lack of Cohesion of Methods (LCOM)	1,7/class	1,7/class	1,8/class	1,1/class Improved
Package Tangle Index (PTI)	11,2%	9,9%	9,6%	14,2% Poor
Dependencies to cut	16 between packages 34 between files	24 between packages 57 between files	32 between packages 74 between files	26 between packages 64 between files
Complexity	1,9/method 8,8/class 8,6/file	1,9/method 9,9/class 10/file	1,8/method 9,8/class 9,8/file	2,1/method 11,0/class 11,2/file
Average Code Coverage	3.0%	0%	0%	Improved

From Table 4.1 above, we can find the actual outcome from Sonar. It is somewhat confusing and does not show the actual improvement in the results, due to the reason that the number of features and functionality for SOLA application was dramatically increased before every release. Therefore, we present graphical representations and interpret the result in ratio for LOC, Comments, Classes and Rules in Figure 4.4 and Response for Classes (RFCs), Lack of cohesion of methods (LCOM) Package Tangled Index (PTI), Dependencies to cut per package and code complexity per class in Figure 4.5.

As we can see, the first version of SOLA contained approximately 30,000 LOC with approximately 24% comments. These codes were integrated within more than 500 classes each of which class has on average of 12 responses each. This gives the ratio of 1:4.1. The rules violated for the first release was approximately 3000 including major and minor violations whereas, for the later versions, these violations were reduced. It can also be seen that the third release of SOLA was huge in terms of LOC. It contained more than 84000 LOC; however the ration for code, comment and violations remained considerable.

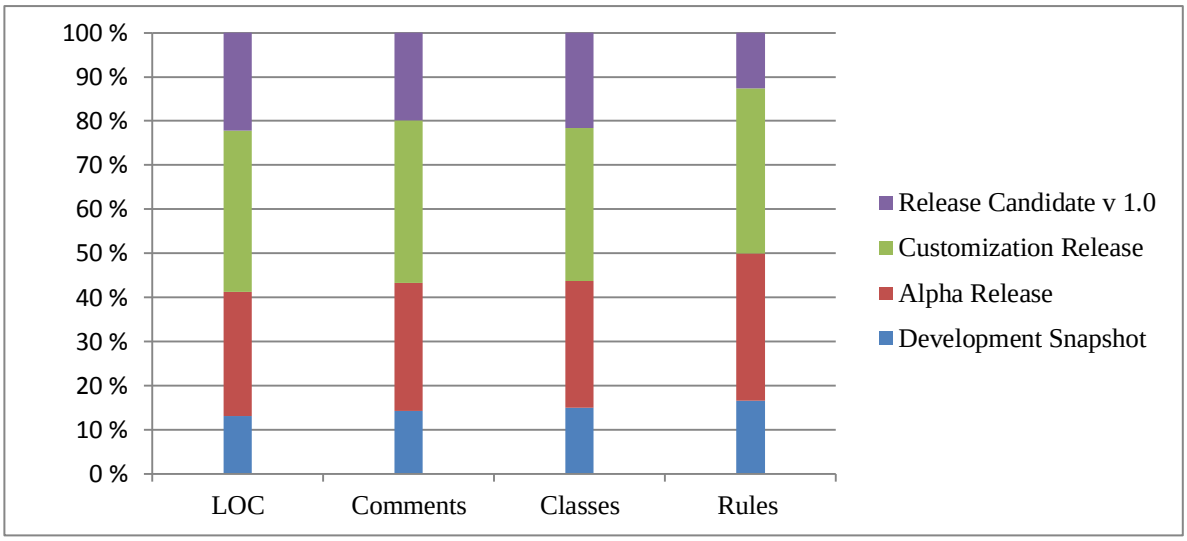


Figure 4.4 Graphical view for the basic metrics

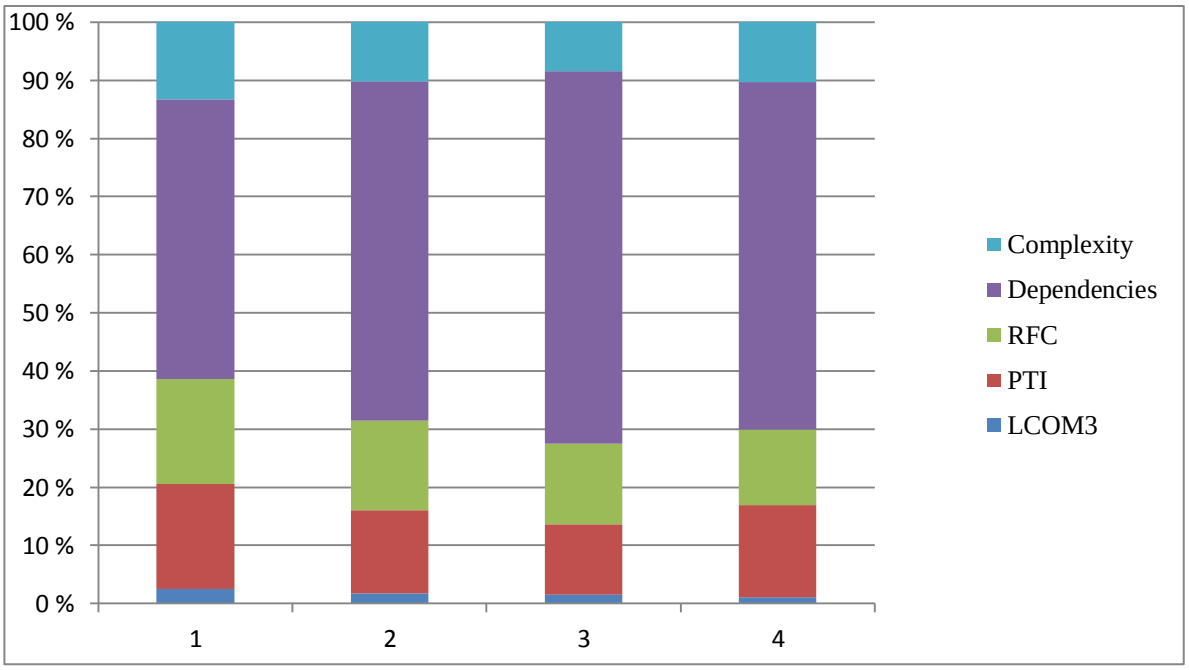


Figure 4.5 Graphical view for Compelxity, Dependencies, RFC, PTI and LCOM3

Similarly for the attributes present in Figure 4.5, we can see that the LCOM3 was significantly improving which showed us that the SOLA was cohesive, the RFC and complexity were improving making the application more maintainable and less tangled for each version resulting in more modular application. The dependencies however were increasing due to more correlated features implemented for later versions. The result after each quality review was either improving or still because of the recommendations made to the previous releases.

Apart from the static review, Table 4.3 below shows some more attributes such as Documentation and supportability for SOLA, Accessibility, Modularity, Security and

Performance which were separately examined for each versions. The results were then reported to the development team. These attributes relates directly to the behavior of the application and are directly related to the number of features and functionalities covered for each version.

4.2.1. Results for Community Compliance Attributes

As mentioned earlier, the results were categorized in two parts, static and behavioral. In the following section the discussion on the behavioral results is made. This behavioral analysis was made with the help of the results of static analysis and executing the application. All the attributes for dynamic or behavioral analysis falls under the category of community compliance.

Behavioral Analysis

Behavioral analysis or the dynamic analysis is the analysis made based on the results from Table 4.1 and also by executing different versions of SOLA application. The results obtained for this section show actual increment in the overall quality of SOLA.

For certain attributes which were reviewed including portability, reliability, usability, maintainability and performance of four different releases, three different machines with different Operating Systems (OS) and system configurations were chosen. Table 4.3 below shows the outcomes.

We can see from Table 4.3 below that the documentation and support i.e. the serviceability measure is found to be up-to-date which means that all the technical and non-technical documents under the category of requirement, architecture, development, deployment, optimization, planning, help and training were separately checked and reported. For example, documents like Statement of Requirement, Use Case Description, Data Model, Architecture documents, Envision Statements, Data Dictionary, Way of Working Document, Communication plans, testing strategy, monthly progress reports and user manual were checked and verified for all the releases, after which “Up-to-date” status was reported for each releases. These documents are available for the public (download via www.flossola.org) making it “Accessible”.

As mentioned earlier, SOLA application was deployed and run in three different machines with different OS resulting in successful deploy and run. However, in Windows Vista the configuration for the application server was time consuming due to the low system configuration. SOLA applications were using different components like virtual machine for the connection to the main server and PGAdmin as the database server. For the reliability review, these components were stopped/ restarted. The responses and behavior of the system were noted. As a result, we found that SOLA is exceptionally reliable in the sense of error handling and periodic backup of the database.

In addition to this, we tried to invade the security of first version of SOLA application which was not possible, but for the third version, when the web services were

added; SOLA failed to check the login to the web services and was easily overrun. The review results were forwarded to the development team.

It was easily understood that the performance is mostly directly affected by the design decisions that has been taken for the software and as we can see in Table 4.3 below, the initial version of SOLA application is listed as a "Low Performance" application because it did not complete on time. The results were sent to the development team who then handled this issue efficiently omitting the flaw for next release. JUnit test cases were prepared and were run through JMeter. Apparently the result hence obtained showed that this software met the requirements set and also the design decisions were effectively chosen. Table 4.2 and Figure 4.6 below are the results and graphs from JMeter for the Customization Release (third release) of SOLA.

Table 4.2 Performance Test (Load test) Result

Requirement	No. of users (threads)	Loop count	Ramp-up Period (s)	Average (ms)	Median (ms)	Expected (sec)
QL-34/35	100	1	3600	184	182	< 5
QL-36				254	249	<5
QL-37				678	621	<8
QL-38	1	-		~6 sec	-	<30

QL in Table 4.2 means the Non-functional requirement sets for the FLOSS SOLA application. Numbers 34 to 38 are the sets for the performance section. [30]

In addition to this, final conclusions for some attributes like serviceability, documentation and more performance test results and graphs from JMeter for the third review are presented in Appendix B.

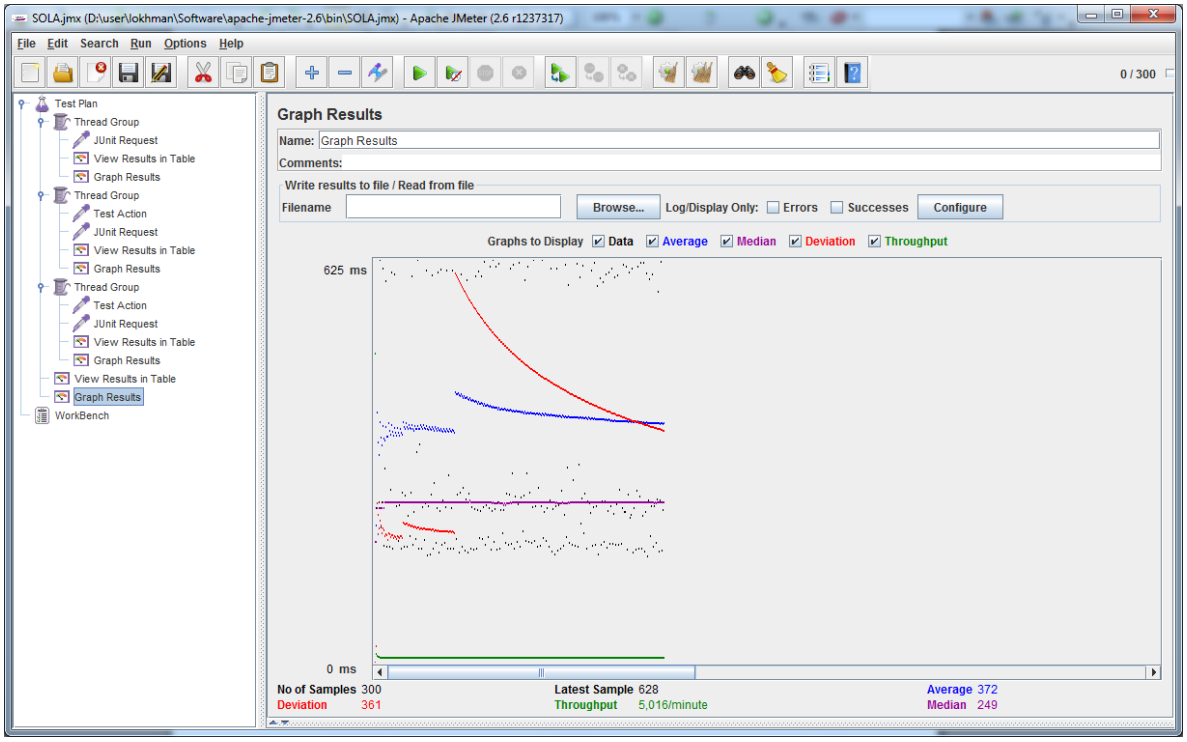


Figure 4.6 General Test of 1 hr with 12 seconds pause using JMeter

As an important quality factor, maintenance was reviewed differently for three different versions of SOLA. In the first version, a possible addition of the language was made with a successful result. For the second version, a calendar was added and for the third version, Maintainability Index (MI) for the entire application was calculated using following formula:

$$\mathrm{MI} = 171 - 5.2 * \ln(\mathrm{V}) - 0.23 * (\mathrm{G}) - 16.2 * \ln(\mathrm{LOC}) + 50 * \sin\left(\sqrt{(2.4 * \mathrm{CM})}\right)$$

Where V is Halstead Volume (1000 assumed),

- G is Cyclomatic Complexity,
- LOC is count of source Lines of Code and
- CM is the percentage of lines of comments.

As a result we got approximately -50 which is in the typical range and was certainly an improvement.

The Usability factor, on the other hand, was not impressive. Because, the features and functionalities covered were increased from 20% to 63% (Table 4.3) for later versions of SOLA application. However, other aspects like the attractiveness, effects and coloring were improved.

The result of Modularity directly depended on the PTI result from Sonar. The result could be seen from Table 4.1 above. The PTI was 11.2% for the first release which was later decreased to 9.9% and 9.6% respectively for 2nd and 3rd releases. For the final release the index was drastically increased to 14% which was certainly an improvement.

Portability was measured by deploying and running the application on two different versions of windows OS including windows vista and windows 7 and Linux Ubuntu 10.04 LTS. The alteration and the behavior of the application was noted and reported. But the final status of 'yes' was due to its ability to accurately and independently function on different platforms and to be accustomed accordingly. Table 4.3 below shows the result from four different reviewed versions of SOLA.

Table 4.3 Comparison result for three different releases (Dynamic)

Metrics/ Attributes	**1st Release**	**2nd Release**	**3rd Release**	**4th Release**
Documentation/ Support	Up-to-date	Up-to-date	Up-to-date	Up-to-date
Accessibility	Yes	Yes	Yes	Yes
Modularity	Less	Improved PTI	Improved PTI	Improved PTI
Portability	Yes	Yes	Yes	Stable
Reliability	No	Error handling and periodic backup was possible.	Stable	Stable
Usability	Issue raised	Issue solved: New Issue raised:	Issue solved: New issue raised:	Issue solved: New issue raised:
Maintainability Index	-	MI = ~83	MI = ~ -50	MI = ~50 – 90
Security	Secured Credential login	Secure	Secure	Secure
Features coverage	~20% NF ~12% F	~50% NF ~20% F	~63% NF ~62% F	~72% NF ~63% F
Performance	Low Performance	Able to handle multiple users in required time.	Met all the requirements from SRS document	Met all the requirements from SRS document

4.2.2. Results for Licensing Compliance Attributes

As a part of licensing compliance check, the licensing compatibility of SOLA to all its related components and its source codes were separately examined. In Table 4.4 below we can see all different SOLA environments that were developed with their respective licenses.

Table 4.4 List of components and their respective licenses [30]

Environment	Licenses
SOLA Client	LGPL, Apache License, BSD, LGPL, CDDL, GPL
SOLA Services	Apache License, LGPL, CPL, BSD, CDDL, GPL
SOLA Database	GPL
Build and Development Environment	CDDL, GPL, Apache License

As SOLA have four major environments that require third party software involvement. All the licenses that were chosen for related components of SOLA are mostly LGPL, Apache, BSD and CDDL/GPL. All of these environments consist of several components. These components, their respective licenses along with the versions of each license are tabled in Appendix C.

Type of licensing scheme that was chosen for SOLA was confirmed for being compatible and was reported accordingly. This was done by analyzing libraries for binary, codes and even by looking for the licensing block on top of all the classes. The first version of SOLA was released using LGPL v2 licensing scheme whereas the later versions were released adopting BSD-3 licensing scheme and was done with required change and compatibility check.

Table 4.5 Licensing Compatibility check for SOLA [31]

	LGPL	GPL	BSD	MPL	CDDL	PHP	Apache	SSPL	Artistic
LGPL	1	1	1	2	2	2	2	1	2
BSD	1	1	1	1	1	1	1	1	1

1: Mixing and linking permissible

2: Only dynamic linking is permissible

As we can see from Table 4.5 above the initial version of SOLA had few restrictions while using LGPL because it allowed only dynamic linking whereas for the later versions of SOLA application the choice of BSD made mixing as well as linking permissible to all of its third party components.

Similarly for reusability, the ease of reusing the application and components was extended deeper towards the code level. Similar to modularity, which covered modular architecture, design and code: reusability concerned to both reusable components as well as reusable codes. Reusability was measured by the count or the ratio of unique methods in a class (more the better) because, with modules containing more unique functions which, if separated, could make the separated code block possible to act individually in other programs.

4.2.3. Results for Method Compliance Attributes

Apart from reviewing the application based on metrics used by Sonar and attributes which act in accordance to the community, we have also reviewed attributes which are especially important and closely visible from the management or administrative perspective (listed in sections 3.2 and 3.3).

For the components that have been used in the development of SOLA application such as Glassfish v3.1 as application server, Netbeans as IDE, PostgreSQL, and PGAdmin III as database server and PostGIS extensions are compatible to the license chosen for SOLA application i.e. BSD 3 Clause. More detail on the list of components, their respective licenses and versions are presented in Table 4.4.

SOLA application has been using 3 layered architecture including presentation layer, service layer and data layer. These layers comprise of several independent components making SOLA architecture reusable in several different contexts.

It has been found that SOLA application and its versions have been accepted and successfully registered in and as F/LOSS product. The product is found to be hosted using a central repository GitHub (github.com/SOLA-FAO). This application has been initiated and developed by UN FAO primarily for developing countries including Nepal, Samoa and Ghana. The interoperability documents by these countries have been successfully reviewed and confirmed as approved.

It has also been noticed that whenever any new tool was used by the core and active developers the required tutorial and training about the chosen infrastructure was provided efficiently which is why there is no ‘alien’ scenario during the development.

Alike all the other products SOLA needed proper marketing for financial assistance as well as community development. In order to do that marketing, research was conducted with all the potential development assistance agencies such as World Bank, ADB, US AID and so on and was efficiently promoted to them via proper communication channels.

A research report was submitted to the project manager and to the team of core developers in UN FAO. The marketing strategy report consists of 7 p’s which are Product, Public License, Place, People, Promotion, Perception and Process. Since the application is a solution to open land administration the promotion were made on land agencies and consultancies, development assistant agencies, academic institutions, social media, open source communities and magazines.

4.3. Discussion

In this section the objective of the thesis is revisited. As mentioned in Section 1.1, the main objective of this thesis was to provide a review framework for OSOS. OSOS are the results caused due to development settings variations from a typical OSS development setting. Alike OSS, OSOS also require review in order to assure its quality. Due to this, the motivations to develop a framework arouse. In order to achieve the objective, a study had to be made in the field of software quality assurance. Study showed that there were many quality assurance models present to ease the vigorous process of software quality assurance.

To choose the best and complete model was a bigger problem. In order ease the effort, 5 most common and well known quality assurance models were chosen. The models are McCall's model, Boehm's model, FURPS framework, ISO/IEC 9126 standard and Garvin's model. These models were then comprehensively analyzed. All of these models were published in late 70's - 80's. It was also found that except FURPS and ISO 9126, rest of the models were product oriented. For example, McCall developed his model concentrating on space and military areas [9].

These models as a whole, does not fit in software developed using open source approach because of mainly two reasons. The first one is that, the idea of OSS was coined later in 80's. Second is that, these models were detailed code leveled. In addition to these reasons, there still remained few other issues why these models could not be adopted for OSS and other software developed using similar development settings. These models mostly lack the fundamental aspects of OSS, for example, accessibility issue which is one essential dimension was missing from these models. In addition, architectural review, product registration and software marketing issues were not properly addressed in these models, frameworks and standards. Despite all drawbacks, we chose these 5 models as the basis of our framework and made them the base models. Due to the reason that we were able to find many product-centric models and frameworks developed lately based on these primitive models. CMM, Dormey, Six Sigma and Prometheus [14] are some examples of the derived work.

In this thesis we extract a set of quality attributes from base models and present them as a suitable framework for reviewing OSOS. These attributes were analyzed and studied individually. We had in our mind three major perspectives to which OSS was viewed namely community, licensing, and development method. This is the reason why LCM is the name we gave to our work.

To mention again, OSOS is also OSS, but with variation in development setting. Therefore, these major perspectives had to be covered in the proposed review framework. All the available quality attributes from the base models were evaluated and interpreted for the context of OSOS. Based on our interpretation, these attributes were then categorized as community, licensing and method compliance attributes. We found 10 relevant attributes from the base models which were Reliability, Maintainability, Performance, Serviceability, Portability, Usability, Modularity, Security, Reusability and Conformance. These attributes were taken from the base models. However, the measures for these attributes were molded and presented to our context. Similarly, we found Accessibility, Software Marketing,

Licensing Compatibility, Trademark, Copyright, Product Registration, and Development Infrastructure equally important for our context but these attributes were missing from the base models. These attributes were then added to the relevant categories in the LCM framework. Most of the attributes that were extracted from the base models fitted well as Community Compliance attributes. It was found that the Licensing perspective and Method perspective were not addressed in the base models.

The LCM framework [Figure 3.10] comprises of 3 major perspectives through which the OSOS could be viewed namely, The Community Compliance, Licensing Compliance and Method Compliance. There are 11 quality attributes as community compliance attributes. Altogether there are 23 sub-attributes used as measures to these 11 attributes. Whereas there are 4 quality attributes as licensing compliance attributes and 2 attributes as Method compliance attributes with no sub-attributes as their measure. All the Licensing and Method compliance attributes are more procedural and less functional.

5. Conclusions

In this chapter we discuss the conclusions made from our result and findings. We also discuss the limitations that were encountered during the research and thesis overall. In addition to this, we propose some ideas for the future development of this thesis, which due to some limitations were not fulfilled.

5.1. Conclusion

The LCM framework was applied to an OSS gave a positive and incremental result. Since this model was applied for reviewing four different versions of the SOLA project which was developed following the open source norms and traditions. Hence we conclude that this model is acceptable to any other software developed using similar approach.

It is an obvious fact that OSS has to go through several reviews also because most of the OSSs are developed following the classical mantra of "Release early, release often", which makes each piece of released version reviewable. It is equally important to know and pick the right review attribute at the right time. There are few review attributes which must be included in the very first release of the software but might not be of equal importance in the later versions. In our case study we came across some situations where the metrics and attributes that were chosen and applied to the first version were no longer used for the later versions, assuming that no additional components (especially organizations) are involved in the later development.

In the context of SOLA review one attribute that was left behind was Conformance. For the first version of SOLA, a licensing compliance attribute Conformance was reviewed which, in later versions, was skipped because the application already met the standards of the OSI, and other related organizations like governmental organizations from three different pilot implementation countries including Samoa, Nepal and Ghana. There were no additional organizations that were involved in the latter development of SOLA. Hence this attribute was found less important. Similarly, Usability and Functionality were, for the first version, a lower priority review attributes, which on the later stage of development, acted as vital ones. On the other hand, some attributes like Reliability and Maintainability were reviewed with the same priority level for each release of the software but with different review scenarios.

5.2. Limitations

There are thousands of review frameworks and QA models available in the web, each model containing several quality attributes. Lot of these attributes remains the same in most of the QA models. Due to which our model has the same trend followed, making our model less different than remaining models, however, we are precisely concerned towards the OSS due to three major perceptions chosen including Open Source as a Community, Open Source as a Licensing Scheme and Open Source as a development method. There are distinct attributes which have a higher degree of impact on all of these approaches, hence making it more modular and usable even if one chose to be focused on a single approach.

5.3. Future Work

Provided enough of time and resource it is possible to extend this thesis, with greater depth towards the method and licensing compliance sections. It is also possible to present, in more detail, the community development and software marketing. The role and perception from administrative level could also be added as the extension to this thesis work. In addition to this, comparison of two or more case studies might lead to precise and more acceptable results. Hence, in the future multiple cases could be considered and result could be made more accurate. The analysis was made to the quality attributes of the base models leaving the sub-attributes and measures alone. In the later work this issue could be addressed in more detail.

References

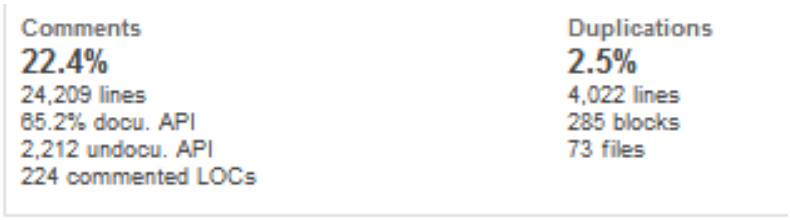
1. Chris DiBona, Sam Ockman, and Mark Stone; *Open Sources: Voices from the Open Source Revolution*, 1st Edition, 1-56592-582-3, January 1999
2. Jesus M. Gonzalez-Barahona, *Free Software / Open Source: Information Society Opportunities for Europe?* v1.2, 24th April 2000
3. Kitchenham, B. and Pfleeger, S. L.; *"Software quality: the elusive target"*, IEEE Software, no. 1, pp. 12-21, 1996
4. Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M., *Characteristics of Software Quality;* North Holland. Pub. Co, New York; 1978
5. A. W. Roscoe Ed; *A Classical Mind: Essays in Honour of C. A. R. Hoare*; Chapter 13, Prentice-Hall 1994.
6. Tobias Otte, Robert Moreton, Heinz D. Knoell; *Development of a Quality Assurance Framework for the Open Source Development Model.* 3rd International Conference on Software Engineering Advances. IEEE, 2008
7. Khashayar Khosravi, Yann- Gael Guéhéneuc; *A Quality Model for Design Patterns*, Technical report 1249, University of Montreal, September 2004.
8. Ronan Fitzpatrick; *Software Quality: Definitions and Strategic Issues*, ITSM, Staffordshire University
9. Ernest Wallmüller; *Software Quality Assurance: A Practical Approach*; Prentice Hall ISBN 0138197806, 9780138197803, 1994
10. Shari Lawrence Pfleeger; *Software Engineering Theory and practice.* Prentice Hall 2001
11. Marc-Alexis Coté, Elli Georgiadou; *Software Quality Model Requirements for Software Quality Engineering*, Software Quality Management and INSPIRE Conference (BSI); 2006
12. Pressman; *Software Engineering: A practitioner's approach*, 5th Edition, Boston: McGraw-Hill, 2001
13. Patrik Berander et al; *Software Quality Attributes and Trade-offs*, Belkinge Institute of Technology, Chapter 1 pp 7, June 2005.
14. Adam Trendowicz, Teade Punter; *Quality Modeling for Software Product Lines*, ECOOP 2003, Germany, July 2003.
15. Rafa E. Al-Qutaish; *Quality Models in Software Engineering Literature: An Analytical and Comparative Study*, Journal of American Science, Al Ain University of Science and Technology, 2010.
16. Eric Steven Raymond, *The Cathedral and the Bazaar*, O'Reilly Media, ISBN: 978-1-56592-724-7 | ISBN 10:1-56592-724-9; 2000.
17. Matthias Sturmer; *Open Source Community Building*, Institute of Information Systems, Bern, March 2005
18. Henri Hedberg et al; *Assuring Quality and Usability in Open Source Software Development*, IEEE, University of Oulu, 2007
19. Morten Sieker Andreasen, Henrik Villemann Nielsen, Simon Ormholt S., Jan Stage; *Usability in Open Source Software Development: Opinions and Practice*, Aalborg

University Denmark, ISSN 1392 – 124X Information Technology and Control, Vol 35, No 3A., 2006
20. Alan McCormack, John Rusnak, Carliss Baldwin; *Exploring the Structure of Complex Software Designs: An Emperical study of Open Source and Proprietary Code*; Working paper, Harvard Business School, June 2005
21. Alessandro Narduzzo, Alessandro Rossi; *Modularity in Action: GNU/Linux and Free/Open Source Development Model Unleashed*; Athesina Studiorum Universitas, May 2003
22. Sonnenburg, Braun, Ong, et al; *The Need for Open Source Software in Machine Learning*; Journal of Machine Learning Research 2443-2466, 2007.
23. Web source at www.apache.org/licenses/GPL-compatibility.html accessed online on 15.07.2012, The Apache Software Foundation webpage. Last modified 2012
24. Web source at www.gnu.org/licenses/license-list.html accessed online on 15.07.2012, GNU Operating System. Free Software Foundation. Last updated 13.07.2012 14:25:52
25. Donald G. Firesmith; *Common concepts underlying safety, security, and survivability engineering*. Carnegie Mellon Software Engineering Institute - Technical Note CMU/SEI-2003-TN-033, December 2003.
26. David A. Garvin; *What does "Product Quality" really mean?* Harvard University, Sloan Management Review, fall 1984.
27. Philip Kotler; *Marketing Management*, 11th Edition, Pearson Education Inc, ISBN 0-13-0497150, 2003
28. I. Stamelos et al; *Code quality analysis in open source software development*; Blackwell Science Ltd., 2002
29. Web source at www.sonarsource.org last accessed on 3.8.2012
30. Web source at www.flossola.org/content/documents SOLA Architecture Document last access on 2.10.2012
31. Web source at tutopen.cs.tut.fi/course13/Open_Source_Introduction.pdf last accessed on 1.04.2013
32. Pfleeger, Charles P.; Pfleeger, Shari Lawrence; *Security in Computing*, 4th Ed.. Prentice Hall PTR. pp. 154–157. ISBN 0-13-239077-9, 2003
33. D. Forrest, C. Jensen, N. Mohan and J. Davidson; *Exploring the Role of Outside Organizations in Free/ Open Source Software Projects*; 8th IFIP WG 2.13 International Conference, OSS 2012.
34. Bugzilla.org contributors; www.bugzilla.org last accessed on 28.03.2013; Last modified February 19, 2013.
35. Web source at www.mantisbt.org last accessed on 28.03.2013.
36. Web source at www.crmonline.com.au/crm/quality-assurance, Overview of CRM Implementation Processes; last accessed on 09.04.2013

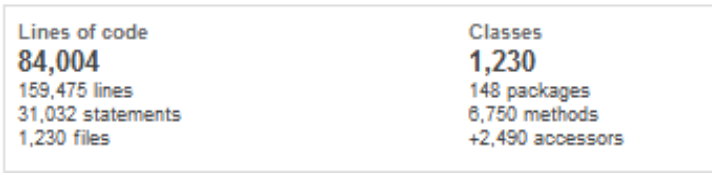
Appendix A

Following are the result snapshots from Sonar for the final version of SOLA application (Customization Release)

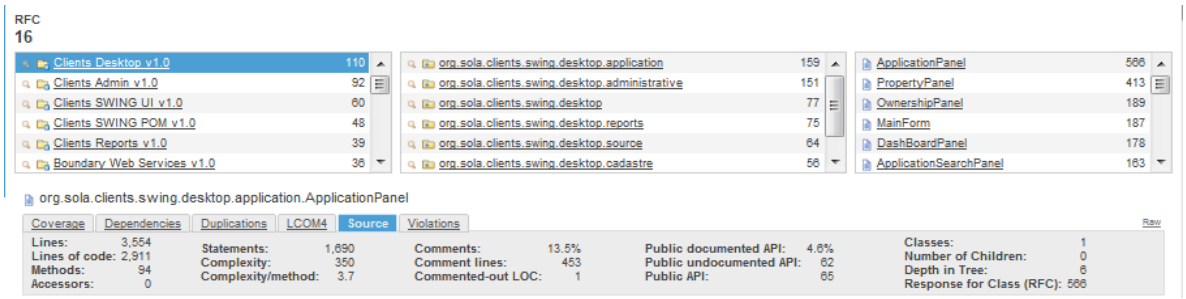
Comments and duplication



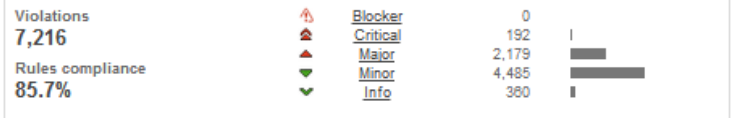
Number of classes and LOC



Response For Classes (sub-project -> packages -> classes) from right to left

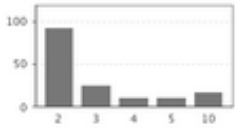


Rules and violations with severity levels



Lack of Cohesion of method

LCOM4
1.8 /class
12.4% files having LCOM4>1



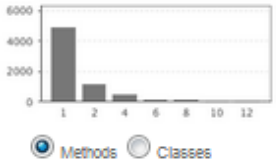
Package Tangle Index

Package tangle index
9.6%
> 62 cycles

Dependencies to cut
32 between packages
74 between files

Complexity

Complexity
1.8 /method
9.8 /class
9.8 /file
Total: 12,092



Code coverage

Coverage
0.9%

Clients POM v1.0	0.7%
Services POM v1.0	1.3%
Boundary POM v1.0	2.2%
Boundary Web Service Clients v1.0	3.3%
EJB Address v1.0	50.0%
Common Messaging v1.0	93.6%

Appendix B

Following are the conclusions drawn from the review made on final version of SOLA application towards serviceability, documentation, and performance test.

1 Serviceability

Serviceability	Results/Conclusion
Help desk notification	N/A
Network monitoring	N/A
Event tracing	N/A
Automated installation packages will be available (QL-12)	One form of SOLA application is available as a web start. This could be considered as an automated installation package.
Email servers for reporting technical errors to support team	N/A
The system will include documentation to support its administration and use. (QL-10)	This feature is available for the current version of SOLA. Documents like user manual, requirement specification documents and read me files could be accessed (downloaded) via FLOSS SOLA web page. www.flossola.org/content/documents
The system will include documentation to support its development, enhancement and maintenance. (QL-11)	This feature is available for the current release of SOLA i.e. Customization release. JavaDocs, Data Dictionary, Architecture Document and developers' wiki is available. www.flossola.org/wiki/Main_Page
Automated testing (QL-13)	FLOSS SOLA is using JUnit application for testing which is an automated testing tool. Hence, this feature is considered to be fulfilled.

2 Documentation progress

Category	Description	Project Documents	Comments
Requirements	Describes the application from the end users perspective and includes; Functional Requirements Specification, Use Case Model, System Vision, System Requirements Specification, screen definitions, Feature List, User Stories, etc.	Statement of Requirements, Use Case Descriptions, FLOSS SOLA Data Model, Software Architecture Document,	*These documents give detailed information on the initial generic software. The architecture document for customization release is yet to be updated (till 18th May 2012)*

		Envision Statement	
Architectural	Describes any constraints imposed on the architecture, rationale behind the architecture and/or the physical structure of the application.	Software Architecture Document, *Data Dictionary is made available for Customization release*	*Deployment Diagram is not available*
Development	Covers detailed system design, development standards and guidelines, code comments, development environment setup procedures and/or development guides.	Software Architecture Documents, Software Review Report, Use Case Description, Way of Working (WoW) Document	All the ideas proposed in WoW documents should be implemented or the WoW document should be revised for next release.
Deployment	Provides details on installation and initial configuration of the application including dependencies with other software components and/or release procedures and release notes.	Readme file, Communication Plan	*The detailed description procedure is available for the customization release.*
Operational	Describes operational procedures and administration tasks to administer and maintain the application.	-	*Not available*
Project and Planning	Provides general information on the project and how it is being managed as well as planning details.	Sola web page, JIRA, Monthly reports	*Detailed plan and updated monthly reports are available*
Testing	Describes how testing will take place, what testing is required and tracks the results of the testing activities.	Quality Plan Document	*This document is not updated for the release of customization release.*
Help and Training	Describes the functions supported by the application and explains how users can interact with it.	Wiki page, SOLA Web page, Training material, user manual, online help	*User manual and online help are available. Training material is not available.*

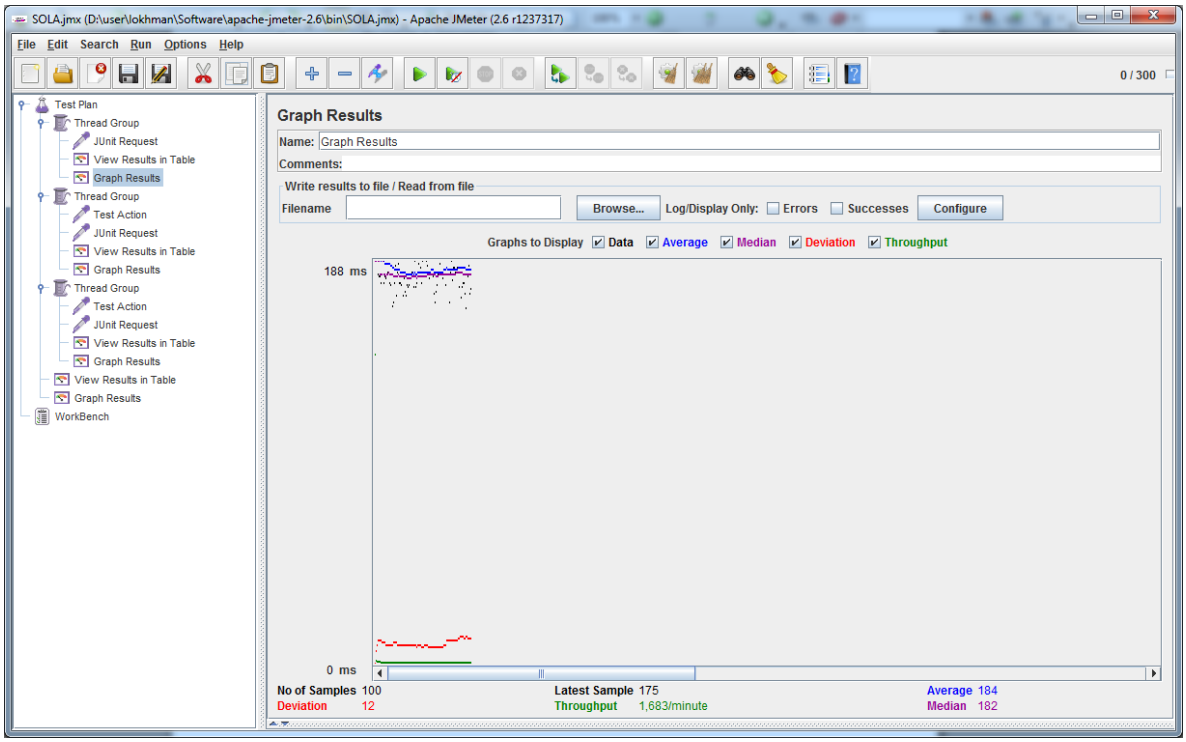


Figure 0.1 Load Test result QL-34/35 -- Test connection to Case Management service with setting user credentials

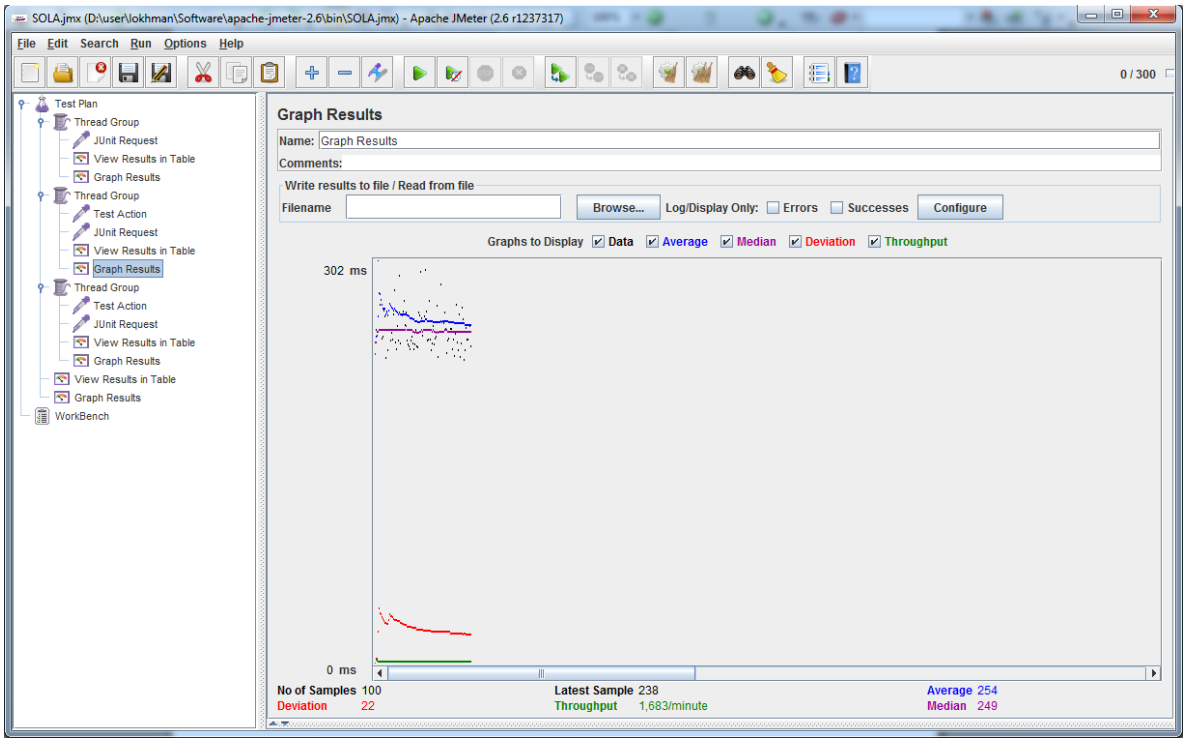


Figure 0.2 Load Test result (WS) for 100 users QL-36 -- Get lists of unassigned and assigned applications from Search service

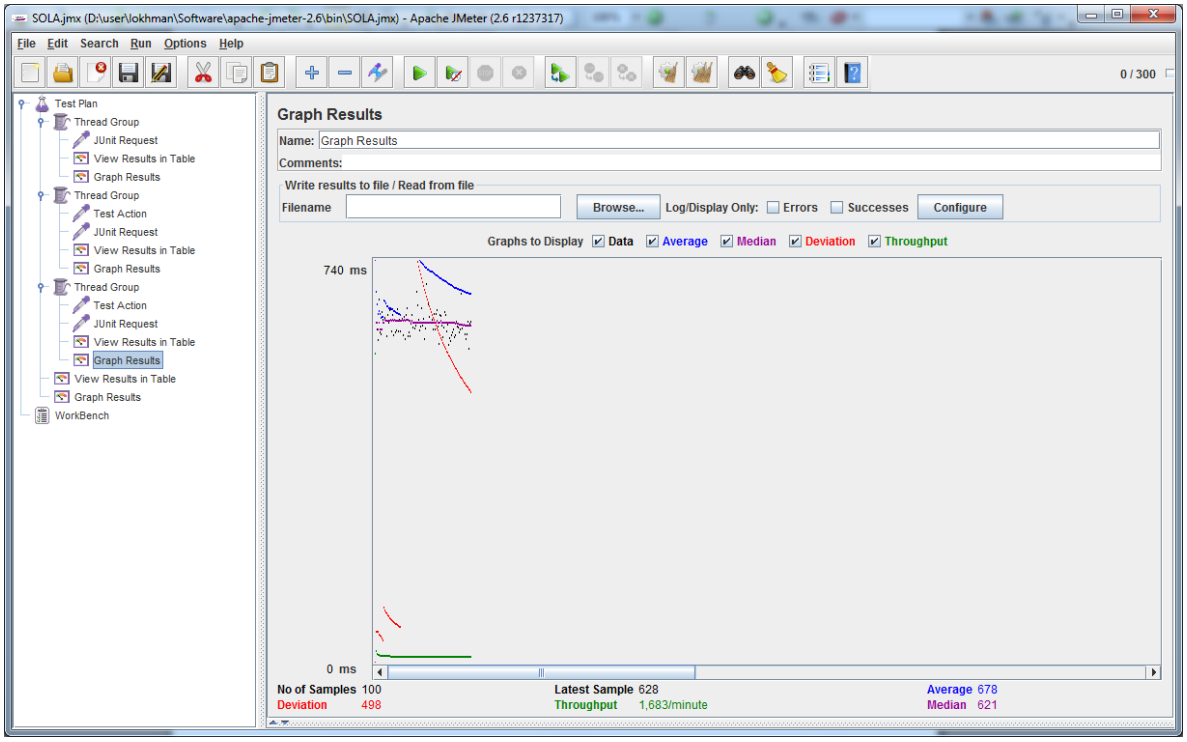


Figure 0.3 Load Test result (graph) for 100 users QL-37 -- Get spatial elements from six different GIS layers

Appendix C

Following is the list of the entire SOLA environment, their components and the corresponding licenses along with the versions.

Environment	Components	License	Version
SOLA Client	Geo Tools	LGPL	2.1
	Jasper Reports		
	Better Bean Binding		
	Swing Labs		
	Barcode4J	Apache License	2.0
	Barbecue Barcode	BSD	3 Clause
	Toedter calendar	LGPL	2.1
	Metro	CDDL/GPL	1.1/2
	Java Application Framework	LGPL	2.1
	JGoodies	BSD	2 Clause
	Map Icons	GPL	2
SOLA Services	Dozer	Apache License	2.0
	Hibernate 3.5	LGPL	2.1
	JBoss Drools	Apache License	2.0
	Imaging Library (JMagick)	LGPL	2.1
	JUnit 4.8	CPL	-
	DateUtility class	LGPL	2.1
	Money class	BSD	3 Clause
	Glassfish (embedded)	CDDL/GPL	1.1/2
	GeoServer	GPL	2
SOLA Database	PostgreSQL license		
	PostGIS	GPL	2
Build & Development Environment	Netbeans	CDDL/GPL	GPL v2
	Maven – glassfish– plugin		
	Maven	Apache License	2.0
	Maven – jarsigner		
	Maven – assembly		

	Jaxwa – maven –plugin		
	Maven – compiler – plugin		
	Maven – war – plugin		
	Maven – ear – plugin		